615

Babán Gábor
Masa István

GÉPI KÓDÚ PROGRAMOZÁS KEZDŐKNEK ÉS HALADÓKNAK

C 16 és PLUS/4 számítógépre

NOVOTRADE

BABÁN GÁBOR – MASA ISTVÁN

1792
196
1988

Gépi kódú programozás kezdőknek és haladóknak

(C 16 és PLUS/4 számítógépekre)

1988 SYS 4126 "HARDWORK"
TOM

1000 4096
101E 30
4126

JSR 1800
JSR ~~FF4F~~ (ESC-N) 1B-4E ([illegible] 90)
TURBO TRANS WITH-LOAD
BY TOT DROT HARDWORX

NOVOTRADE RT. 1988

Lektorálta: TORY KÁLMÁN

MÁSODIK KIADÁS

A kiadásért felel: RÉNYI GÁBOR a Novotrade Rt. igazgatója
Felelős szerkesztő: Siba László
Műszaki szerkesztő: Erdősi Zoltán
ISBN 963 02 4872 7

Készült a Sylvester János Nyomdában, Szombathely
Felelős vezető: Hanuszek Béla igazgató

TARTALOMJEGYZÉK

FÜGGELÉK

ELŐSZÓ

Könyvünk élesen elkülöníthető két fő részből áll. Az első fejezet lényegében tankönyv. Azok számára készült, akik a gépi kódú programozást kívánják – esetleg minden előismeret nélkül – megtanulni. Erre egyre többen vállalkoznak, de megfelelő felépítésű és részletességű könyv, ill. sikerélmények hiányában sokan feladják. Ez a fejezet részletességével és közérthető nyelvezetével biztos támaszt nyújt a gépi kódú programozás kezdő lépéseihez. Ráadásul nem csak a C 16-os és Plus/4-es tulajdonosoknak szól, hanem bizalommal forgathatják a C 64-es, C 128-as, C 610-es tulajdonosok is, mivel ezeknek a gépeknek az utasításkészlete megegyezik az előzőekével.

A második fejezet azok számára készült, akik a programozási alapismereteket megszerezték. Itt már az volt a cél, hogy minél több információt nyújtsunk magáról a C 16-os és a Plus/4-es típusú gépekről, a nem feltétlenül gépi nyelven programozó Olvasó számára is. Törekedtünk arra, hogy lehetőleg teljes tájékoztatást nyújtsunk a gépek programozási lehetőségeiről, az operációs rendszer és a BASIC-interpreter működéséről, valamint bizonyos programozási fogásokról. A perifériák kezeléséről szóló alfejezet sem csak a C 16-os és Plus/4-es tulajdonosoknak szól, minden korlátozás nélkül használhatják a C 64-esen programozók is.

Könyvünk megírásakor számítottunk arra, hogy az Olvasó a számítógép mellett tanul, és a példákat rögtön ki is próbálja. Véleményünk szerint ez a tanulás egyik leghatékonyabb és azonnali sikerélményeivel legösztönzőbb formája. Reméljük, hogy könyvünk sikeresen betölti azt az űrt, ami a számos BASIC tankönyv mellett a gépi kódú programozást oktató könyvek terén mutatkozik.

1. FEJEZET
GÉPI KÓDÚ PROGRAMOZÁS KEZDŐKNEK

1.1. A BASIC és a gépi kódú programozás

Miért tanuljuk meg a gépi nyelvet?

A C 16-os és a Plus/4-es számítógép – hasonlóan az összes nálunk forgalomban lévő mikrogéphez – BASIC nyelven programozható. Ezt a nyelvet már viszonylag sokan elsajátították, vagy éppen most tanulják. Az ezeken a gépeken megvalósított BASIC, a 3.5-ös verzió jelentős előrelépést jelent a C 64-es BASIC-jéhez képest. Egyszerűen megvalósítható vele a legtöbb programozási feladat.

A BASIC nagyon könnyen megtanulható, kialakításakor ez volt a legfőbb cél. Ezekután felmerülhet az a kérdés: miért kellene megtanulni gépi nyelven programozni?

Sok érvet lehetne felhozni, de a legdöntőbb a programvégrehajtási sebesség. Az azonos funkciójú gépi nyelven írt program a BASIC-változathoz képest nagyságrendekkel gyorsabb, a program jellegétől függően. (Találomra kiválasztott tesztprogramunknál ez a sebességnövekedés 416-szoros volt.) Egyszerűbb programoknál mindennek semmi jelentősége sincs, de gyakran előfordul, hogy a BASIC lassúsága miatt alkalmatlan az adott feladat megoldására. Játékoknál, valamint általában akkor, ha a képernyőn gyorsan mozgó figurákat szeretnénk megjeleníteni, szín- és hangeffektusokat használni, ez mindig fennáll. Mivel hobbigépről van szó, más alkalmazás szinte alig fordul elő. A legtöbb kezdő programozó csodálkozva és irigykedve nézi a játékprogramok lenyűgöző megoldásait. Ebbe a világba pedig a gépi kódú programozás megtanulásán keresztül vezet az út.

A gépi nyelvet ismerők világa sokáig elég zárt volt – nem véletlenül – hiszen az a tévhit uralkodott, hogy a nyelv megtanulása óriási szakmai felkészültséget, és rendkívüli erőfeszítést igényel. Az olcsó mikrogépek elterjedésével mindez pillanatok alatt szertefoszlott. Jellemző a helyzetre, hogy a legújabb hobbi célú gépekbe, mint például a C 16-osba és Plus/4-esbe a gyártók gépi programozást segítő monitorprogramot építettek be, szakítva ezzel korábbi koncepciójukkal.

Mi is az a gépi kód?

Önmagában egyetlen számítógép sem érthet meg egy olyan magasszintű programnyelvet, mint a FORTRAN, a PASCAL, a FORTH, és természetesen a BASIC is. Ahhoz, hogy egy gépet BASIC nyelven programozni lehessen, nem elég önmagában a számítógép gépi része. Szükség van egy programra, amely a *tolmácsolást* végzi a programozó által használt BASIC, és a mikroprocesszor saját nyelve között. Az pedig talán magától értetődik, hogy ezt a *tolmácsprogramot,*

amelyet INTERPRETER-nek neveznek, kizárólag a processzor saját nyelvén, tehát gépi kódban írhatták meg. Ez a program a gép ROM-jában (Read Only Memory), tehát a csak olvasható és kitörölhetetlen tárban van és azonnal elindul, ha a gépet bekapcsoljuk. Ez azonban a BASIC nyelven programozó számára nem látható, ő csak azt érzékeli, hogy a gépe megérti és végrehajtja a BASIC-et. Abban a pillanatban azonban, ha különösebb, az alapszavakkal nem végrehajtható feladattal áll szemben, egyetlen lépést sem tehet a gépi nyelv ismerete nélkül.

Legfőbb jellemzőinek leírása előtt azonban térjünk ki néhány alapfogalomra. Lehet, hogy ezek a legtöbb BASIC-ben programozó számára is ismertek, de a teljesség kedvéért nem hagyhatjuk ki.

Minden számítógéptár rekeszekből áll, ezekbe különböző számokat írhatunk, és később vissza is olvashatunk. Ezt a típusú tárat RAM-nak (Random Access Memory) nevezik. Van ezenkívül minden számítógépben ROM is, tehát csak olvasható, nem törölhető tár is a rendszerprogramok és a rendszer állandó adatai számára.

Azért, hogy ezeket a rekeszeket meg lehessen különböztetni egymástól, számokkal látták el őket. Ezt a számot az illető tárrekesz *címének* nevezik, azt a számot pedig, amit kiolvasunk belőle, az adott cím tartalmának. A gépi kódú programok ezekbe a rekeszekbe kerülnek be.

A gépi nyelv – hasonlóan a BASIC-hez – utasításokra épül. Abban is hasonlítanak, hogy az utasításszót legtöbbször követi valami (az operandus), ami megmondja, hogy az illető műveletet mivel vagy hogyan hajtsa végre a gép. Mivel a BASIC-et a legtöbb olvasó valószínűleg ismeri, könyvünkben kiemeljük, hogy az éppen tárgyalt utasításhoz a BASIC-ben milyen utasítás hasonlít a legjobban.

Egy gépi utasítás egy tárrekeszt foglal el. Ha van operandus, az a következő egy vagy két rekeszben van tárolva. A következő rekeszben értelemszerűen a következő gépi utasítás kódja van. Íly módon egy gépi program nem más, mint egy tárterület, melyben az egymást követő címekre bizonyos értékek vannak beírva. Ebben a formában tehát egy hatalmas, áttekinthetetlen számhalmaz.

Szerencsére a helyzet nem ilyen kétségbeejtő. Már a számítástechnika kezdetén feltalálták az emlékeztető kódokat (mnemonik), tehát minden gépi utasításnak nevet adtak. Esetünkben minden ilyen név három betűből áll, ami egyértelműen utal az utasítás jellegére. Ennélfogva igen könnyen megtanulható. A gépi nyelven programozó, aki monitort vagy assemblert használ, kizárólag ezekkel a mnemonikokkal fog csak találkozni. Amikor egy tárterületet listáz (disassemblál), akkor soronként egy mnemonik fog megjelenni a hozzá tartozó operandussal. Ez már hasonlít a magasszintű nyelvek programsoraihoz. Ezek után semmi nem indokolja a gépi programozástól való idegenkedést.

A gépi nyelv utasításai sokkal elemibb műveleteket hajtanak végre, mint azt a magasszintű nyelveknél megszoktuk. Egy-egy BASIC utasítás hatására hatalmas gépi programokat hajt végre a gép. Emellett bizonyára meglepő, hogy egy szokványos feladat BASIC megvalósítása nagyobb tárigényű, mint a gépi kódú változat.

1.2. A kettes és a tizenhatos számrendszer

A köznapi életben a tízes számrendszert alkalmazzuk, a számítástechnikában ezzel szemben a kettes és tizenhatos számrendszer a használatos. Ha tehát programozni szeretnénk, elengedhetetlen e számrendszerek alapos ismerete. Mielőtt azonban elkezdenénk részletezni a kettes és tizenhatos számrendszert, próbáljunk a tízes számrendszerből kiindulva néhány általános megállapítást tenni, ami minden számrendszerre igaz.

A tízes számrendszer alapszáma a 10-es azért, mert a számrendszer helyi értékeinek alapjául szolgál. Minden számrendszer helyi értékei az alapszám 0-tól egyesével növekvő hatványai. A tízes számrendszernél ez a következőképpen néz ki:

10 ↑ 0 = 1
10 ↑ 1 = 10
10 ↑ 2 = 100
10 ↑ 3 = 1000
10 ↑ 4 = 10000
10 ↑ 5 = 100000
...stb.

Ha már tisztában vagyunk a helyi értékekkel, akkor nézzük a következő fontos alkotórészt, magát a számjegyeket. A tízes számrendszer legnagyobb számjegye a 9-es, számjegyei tehát 0-tól 9-ig terjednek. Ebből levonhatjuk azt a lényeges következtetést, hogy minden számrendszer számjegyei 0-tól az alapszámnál eggyel kisebb számig terjednek. (Az alapszám a 10-es, a legnagyobb számjegy a 9-es.) Ennek nagyon egyszerű magyarázata van. A 9-es után, ami 9 db egyes, nem 10 db egyes jön, hanem egy db tízes. Ugyanígy a 90 után – ami 9 db tízes – nem tíz db tízes jön, hanem 1 db százas. Tehát, ha egy számjegy eléri egy helyi értéken az alapszámot, akkor ezen a helyi értéken 0 lesz, a következő, eggyel felette lévő helyi értéken pedig eggyel több. Ha ott 0 volt, akkor 1 lesz, ha 1 volt, akkor 2 stb.

Nézzünk most egy példát, ami egy szokványos tízes számrendszerbeli szám ilyen felbontását mutatja helyi értékekre és a helyi értékeken álló számjegyekre:

1986

1*1000=1000
9* 100= 900
8* 10= 80
6* 1= 6
―――
1986

10↑3	10↑2	10↑1	10↑0
1000	100	10	1
1	9	8	6

Ezután próbáljuk a meglévő ismereteink alapján felépíteni a kettes számrendszert. Alapszáma tehát a 2. Ebből következnek a helyi értékek. Ezek a 2-es szám 0-tól egyesével növekvő hatványai:

$2 \uparrow 0 = 1$
$2 \uparrow 1 = 2$
$2 \uparrow 2 = 4$
$2 \uparrow 3 = 8$
$2 \uparrow 4 = 16$
$2 \uparrow 5 = 32$
$2 \uparrow 6 = 64$
$2 \uparrow 7 = 128$
$2 \uparrow 8 = 256$
...stb.

A legnagyobb számjegy az alapszámnál eggyel kisebb, ami az 1-es. A számjegyek tehát 0-tól az alapszámnál eggyel kisebb számig terjednek, ezek a 0 és az 1. Most, hogy ismerjük a 2-es számrendszer számjegyeit és helyi értékeit, leírhatjuk vele az első számunkat.

Az átváltáshoz használt eljárás nem csak a kettes számrendszerbe történő átváltásnál alkalmazható. Megértéséhez váltsuk át az 1986-ot kettes számrendszerbe.

Az átváltandó számot először osszuk el kettővel, majd írjuk le az eredményt és a maradékot. A maradék 0, ez a kettes számrendszerbeli szám utolsó számjegye. Az osztás eredménye 993, ezt kell ezután tovább osztani és a maradékokat jobbról balra haladva egymás mellé írni. Az osztást akkor kell befejezni, ha az eredmény 0.

eredmény	maradék		
1986	0	(1986:2=993,	maradék=0)
993	1	(993:2=496,	maradék=1)
496	0	(496:2=248,	maradék=0)
248	0	(248:2=124,	maradék=0)
124	0	(124:2=62,	maradék=0)
62	0	(62:2=31,	maradék=0)
31	1	(31:2=15,	maradék=1)
15	1	(15:2=7,	maradék=1)
7	1	(7:2=3,	maradék=1)
3	1	(3:2=1,	maradék=1)
1	1	(1:2=0,	maradék=1)
0	vége		

A tízes számrendszerbeli 1986 a kettes számrendszerben leírva 11111000010.

Végezzük el az ellenőrzést is, ami a helyiértékek és az ott álló számjegyek szorzatainak az összegzése. Kezdjük el balról jobbra:

1 db	1024-es–1	* 1024	=	1024
1 db	512-es–1	* 512	=	512
1 db	256-os–1	* 256	=	256
1 db	128-as–1	* 128	=	128
1 db	64-es–1	* 64	=	64
0 db	32-es–0	* 32	=	0
0 db	16-os–0	* 16	=	0
0 db	8-as–0	* 8	=	0
0 db	4-es–0	* 4	=	0
1 db	2-es–1	* 2	=	2
0 db	1-es–0	* 1	=	0
				1986

A kettes számrendszerbeli számot az eléírt százalékjellel (%) szoktuk megkülönböztetni (pl.%11111000010). Most már elvégeztük a tízesből kettesbe, az ellenőrzés során pedig a kettesből tízesbe való átalakítást is. Javasoljuk, hogy még néhány egyéni példával folytassuk a gyakorlást. Azért lényeges, hogy a kettes számrendszerben otthonosan mozogjunk, mert a számítógép működése ezen a számrendszeren alapul.

A kettes számrendszerben egy számjegy az a legkisebb egység, amit a számítógép tárjának elemi része tárolni képes. Ezt az elemi, legkisebb egységet nevezzük BIT-nek. Ha egy bit értéke 0, akkor kikapcsoltnak (alacsony állapot), ha 1, akkor bekapcsoltnak (magas állapot) nevezzük. Egy 8 bites csoportot nevezünk BYTE-nak. A számítógép processzora és tára is ezekre a 8 bites csoportokra épül, azaz byte-szervezésű.

Egy byte-on belül a 8 bitet jobbról balra 0-tól 7-ig terjedő sorszámokkal látjuk el. Eszerint a legkisebb helyi értékű, a 0. bit a jobb szélső, míg a legnagyobb helyi értékű, a 7. bit a bal szélső.

Ezek után térjünk rá a tizenhatos, más néven hexadecimális számrendszerre. Próbáljuk ugyanúgy lépésről lépésre felépíteni, mint a kettes számrendszert. Annyit rögtön tudunk, hogy az alapszám a 16-os, a helyi értékek pedig a 16-osnak 0-tól egyesével növekvő hatványai:

16 ↑ 0= 1
16 ↑ 1= 16
16 ↑ 2= 256
16 ↑ 3=4096
...stb.

Ezután határozzuk meg a számjegyeket. A számjegyek 0-tól az alapszámnál, tehát a 16-osnál eggyel kisebb számig, a 15-ösig terjednek. Itt jelentkezik az első furcsaság. Mi, akik a tízes számrendszerhez vagyunk szokva, legnagyobb számjegyként a 9-est ismerjük. Itt pedig még a 15-ösig önálló számjegyekre van szükség. Erre a hiányzó hat számjegyre vezették be az ABC első hat betűjét A-tól F-ig.

DEC		HEX		BIN
1	=	1	=	0001
2	=	2	=	0010
3	=	3	=	0011
4	=	4	=	0100
5	=	5	=	0101
6	=	6	=	0110
7	=	7	=	0111
8	=	8	=	1000
9	=	9	=	1001
10	=	A	=	1010
11	=	B	=	1011
12	=	C	=	1100
13	=	D	=	1101
14	=	E	=	1110
15	=	F	=	1111

Az 1986 tehát a tizenhatos számrendszerben:

16↑2	16↑1	16↑0
256	16	1
7	C	2

Végezzük el az ellenőrzést (C=12):

2 db	1-es–	2* 1 =	2
C db	16-os–	12* 16 =	192
7 db	256-os–	7*256 =	1792
			1986

A tizenhatos számrendszerbeli számok leírásánál a szám elé dollár ($) jelet kell írni, ezzel jelöljük, hogy a szám hexadecimális (Pl. $7C2). Ezzel megismertük a tizenhatos számrendszert, bár néhány példával nem árt még gyakorolni az átalakítást tízesből tizenhatosba és vissza.

Ezután térjünk rá, hogyan lehet kettesből tizenhatosba, ill tizenhatosból kettesbe átalakítani a számokat. Maradjunk az eredeti példánknál, és az 1986-os számot kettesből alakítsuk át tizenhatosba. Először írjuk le a kettes számrendszerbeli számot,

11111000010

majd jobbról balra osszuk fel négyes csoportokra. Ha nem jön ki négy számjegyre, egészítsük ki 0-kal:

0111,1100,0010

Ha kész, mindegyik négyes csoportot váltsunk át külön-külön tízes számrendszerbeli számmá, majd írjuk le az így kapott számot tizenhatos számrendszerbeli számjegyekkel:

0111,	1100,	0010,
7	12	2
7	C	2

A végeredmény az 1986 tizenhatosbeli alakja, a $7C2. Végezzük most el visszafelé a műveletet, a $7C2-t írjuk át kettes számrendszerbeli számmá. Ehhez szintén először írjuk le a 7C2 számjegyeket, majd mindegyik számjegyhez rendeljünk hozzá egy négybites csoportot. Ezekben a négyes csoportokban írjuk majd le a hozzá tartozó tizenhatos számrendszerbeli számjegyet kettes számrendszerben. Ügyeljünk arra, hogy minden helyiértéket kitöltsünk, írjuk le az összes 0-t is.

7	C	2
0111,	1100,	0010

Ha a számsor bal oldaláról elhagyjuk a felesleges nullákat, megkapjuk az eredeti kettes számrendszerben leírt 1986-os számot.

$7C2 =%11111000010 = 1986

1.3. A 8501-es processzor regiszterei

A gépi kódú programunk, melyet majdan írni fogunk, kivétel nélkül a 8501-es processzor regiszterei és a tár között fog valamilyen adatátvitelt, adatmódosítást végezni. Ezért elkerülhetetlen, hogy a processzor belső regisztereit tanulmányozzuk, működésüket, tulajdonságaikat, lehetőségeiket pontosan megismerjük.

Ha a BASIC-ben kiadjuk a MONITOR parancsot, vagy a monitor **R** parancsát hajtatjuk végre, akkor megjelenik a képernyőn a processzor regisztereinek tartalma. A következőkben ismertetjük a regiszterek szerepét és lehetőségeit.

Akkumulátor (AC)

Ez a processzor talán legfontosabb, legtöbbet használt regisztere. A gépi utasítások az akkumulátoron keresztül valósítják meg az adatforgalmat. Ide lehet betölteni egy tárcím tartalmát, innen lehet kiírni az adatot bármely tárcímre. A leglényegesebb tulajdonsága pedig, hogy ide kell tölteni azt az értéket, amivel valamilyen aritmetikai vagy logikai műveletet szeretnénk végezni, valamint az eredmény is itt keletkezik.

X-regiszter (XR)

Az adatforgalomban nincs olyan nagy szerepe, mint az akkumulátornak, bár tud néhány olyan adatkezelő utasítást, amit az akkumulátor. Van azonban egy fontos tulajdonsága, ami miatt indexregiszternek is szokták nevezni. Bizonyos címzési módoknál (amiket később fogunk megismerni), indexként használják. Ez a táblázatok kezelésénél nyújthat nagy segítséget.

Y-regiszter (YR)

Funkciójában megegyezik az X-regiszterrel, van azonban néhány olyan címzési mód, amelyet csak az Y-regiszterrel lehet megvalósítani.

Állapotregiszter (SR)

Ez a regiszter 8 jelző- ill. kapcsolóbitet tartalmaz, amiből csak 7 kihasznált. Az 5. bit értéke mindig 1. Öt jelzőbit ad tájékoztatást a végrehajtott műveletek és utasítások eredményéről, két kapcsolóbit pedig a processzor működését befolyásolja. A hét bitnek külön neve van, összefüggésben a funkciójával.

Az állapotregiszter felépítése a következő:

7	6	5	4	3	2	1	0
N	V	1	B	D	I	Z	C

N – NEGATÍV jelzőbit. Értéke akkor egy, ha egy művelet vagy utasítás eredménye nagyobb, mint 127. Értelmezhetjük úgy is, hogy az eredmény legfelső, azaz a 7. bitje egy előjelbit. Ha értéke 1 (a szám nagyobb, mint 127), akkor az eredményt negatív számként értelmezzük, 0 esetén pedig pozitívnak. Az N-bit tehát az eredmény 7. bitjét figyeli, s azzal mindig azonos (BMI, BPL utasítások).

V – OVERFLOW (túlcsordulás) jelzőbit, jelzi a belső túlcsordulást. Értéke akkor lesz 1, ha a művelet eredménye a később ismertetésre kerülő, előjeles kettes komplemens alakban nem fér el az akkumulátorban (BVS, BVC utasítások).

B – BREAK jelzőbit. Értéke akkor lesz magas (1), ha egy BRK utasítás hajtódik végre. Arra szolgál, hogy meg lehessen különböztetni a BRK utasítást és a hardver által kiváltott megszakítást. E kettő megkülönböztetésére a BRK-nál a B-bit 1 lesz.

D – DECIMÁLIS számítási mód kapcsolója. Ha alacsony (0), akkor a processzor az aritmetikai utasítások végrehajtásakor az adatokat kettes számrendszerbeli számokként kezeli.

D = 0 esetén:

$$\begin{array}{rr} & 00001110 = 14 \\ + & 00100111 = 39 \\ \hline & 00110101 = 53 \end{array}$$

Ha a D-bit 1, akkor a processzor az adatokat BCD (Binary Coded Decimal) számokként kezeli. Ekkor az adatbyte alsó és felső 4–4 bitjét egy-egy decimális számnak tekinti és így végezi el a műveletet.

$$\begin{array}{rr} & 0001{,}0100 = 14 \\ + & 0011{,}1001 = 39 \\ \hline & 0101{,}0011 = 53 \end{array}$$

I – INTERRUPT (megszakítás) kapcsoló. Magasra állításával letiltjuk a megszakítást, törlésével pedig engedélyezzük.

Z – ZERO jelzőbit. Akkor lesz magas, ha egy művelet vagy utasítás eredménye 0 volt, egyébként alacsony.

C – CARRY átvitel jelzőbit. Ha pl. egy összeadás során az eredmény nagyobb lesz mint 255, akkor a C-bit 1 lesz. Bizonyos értelemben felfogható az akkumulátor 9. bitjének. Az eltoló utasításoknál a kicsorduló bitek értékét tárolja.

Verem, veremmutató

A verem (Stack) olyan átmeneti adattár, amelyet a programozó és a processzor egyaránt igénybe vehet. Ide be lehet írni és ki lehet olvasni adatokat. Területe $100-tól $1FF-ig terjed. Felépítésének szemléltetésére képzeljünk el egy alulról zárt csövet. Ebbe egymás után lehet betenni az adatokat, egyiket a másik fölé. Ha ki akarjuk venni azokat, mindig a legfelsőt, tehát az utoljára betett adatot tudjuk először kivenni. Ez a LIFO (Last In, First Out) tárolási mód. A verem kezelését könnyíti meg az ún. veremmutató (SP – Stack Pointer). Ez a mutató mindig a verem soron következő üres helyére mutat. A verem hosszából következik, hogy a veremmutató értéke $00-tól $FF-ig terjedhet (ez természetesen a $100-tól $1FF-ig terjedő értékeket jelenti). A verembe íráskor tehát a veremmutató által meghatározott helyre íródik be az adat és a veremmutató értéke (automatikusan) eggyel csökken. A kiolvasás során a veremmutató értéke eggyel növekszik. Úgy is lehet fogalmazni, hogy a veremmutató mindig megmutatja, hány üres hely van a veremben. Ha pl. a veremmutató értéke $FF, akkor még 255 üres hely van a veremben, és az első beírás a $1FF-re történik. Ha a sorozatos beírások után a mutató értéke lecsökken $80-ra, akkor már csak 128 üres hely van a veremben, és a következő beírás a $180-as veremcímre történik.

Utasításszámláló (PC)

Az egyetlen olyan regiszter, mely 16 bitesként kezelt. Tulajdonképpen nem igazán 16 bites regiszter, csak két 8 bites regiszter van összekapcsolva alsó és felső byte-ként. Ebben a regiszterben van az éppen végrehajtás alatt álló gépi kódú utasítás kezdőcíme. Ez lehet az általunk írt program, lehet valamelyik rendszerprogram is a ROM-ból, de az egyik biztosan. Hiszen vagy az általunk írt program,

vagy az operációs rendszer programja mindig fut a gépben, s ebből következik, hogy az utasításszámláló tartalma is mindig változik. A MONITOR R parancsával kiíratva az utasításszámláló (PC – Program Counter) tartalmát, az a parancs kiadása előtt használt utolsó értéket mutatja.

1.4. Utasítások, címzési módok

1.4.1. ÉRTÉKADÁSOK

A BASIC-ben többféle értékadás létezik. Az egész, a valós, és a szöveges típusú változók értéket kaphatnak, és értéküket (tartalmukat) átadhatják más változóknak is. Emellett a tárhelyek is működhetnek hasonlóképpen a POKE és PEEK utasítások segítségével.

```
A  = 1
B  = A
E  = PEEK(43)
D$ = "KELEMEN"
```

Gépi programoknál nincs ilyen változatosság. Értéket csak az egyes tárrekeszek és a regiszterek kaphatnak és adhatnak. A tárhelyeket a címük azonosítja, a regiszterek pedig egyediek, a nevük egyértelműen meghatározza őket.

Nézzünk példát az értékadásra:

```
LDA #$01
```

Jelentése: az akkumulátor tartalma legyen egyenlő 1-gyel.

Az értékadásban szereplő $01 érték természetesen szabadon változhat $00 és $FF határok között, céljainknak megfelelően.

A fenti utasítással analóg BASIC utasítás:

```
A=1
```

Itt azonban a jobb oldali érték sokkal tágabb határok között változhat.

Hasonló utasítás az

```
LDX #$28
```

azzal a különbséggel, hogy az X-regiszterbe tölti a $28 hexadecimális számot.

Az Y regiszterbe töltő utasítás például:

```
LDY #$08
```

amely 8-at tölt az Y regiszterbe.

Az értékadások egy gyakoribb fajtája az, amikor a regiszter egy tárrekesz tartalmát veszi át.

```
LDA $FF19
```

BASIC hasonmása az A = PEEK(65305). Ez az utasítás az akkumulátorba tölti azt a számot, amit a $FF19-es (65305) tárrekesz tartalmaz. (Ebben a rekeszben a keretszínre vonatkozó szám van, bekapcsolás után $EE az értéke.)

A $ jelet itt 0-tól FFFF-ig bármilyen hexadecimális szám követheti.

Az X-regiszter egy hasonló utasítása:

```
LDX $FF15
```

Itt az X-regiszterbe kerül a $FF15-ös rekesz tartalma. (Ebben a rekeszben a háttérszínre vonatkozó szám van, bekapcsolás után $F1 az értéke.) Az Y-regiszter megfelelő utasítása:

```
LDY $FF15
```

Fontos megjegyezni, hogy egy regiszterre vonatkozó értékadással csak az adott regiszter értéke változik, a többi – egy kivételével – változatlan marad. Ez a kivétel az állapotregiszter, amelynek bizonyos bitjei majdnem minden utasítás alkalmával átállítódnak.

```
LDA #$00
```

a Z jelzőbitet 1-re, az N-et pedig 0-ra állítja, miközben kinullázza az akkumulátort.

```
LDY #$FF
```

a Z jelzőbitet 0-ra, az N-et pedig 1-re állítja, eközben az Y-regiszter tartalma pedig $FF lesz.

Ezek után azt nézzük meg, hogy hogyan lehet egy tárhelyre tetszőleges értéket beírni.

```
STA $0C00
```

Az utasítás hatására az akkumulátor tartalma bemásolódik a $0C00-ás rekeszbe, miközben (természetesen) az akkumulátor értéke nem változik. Az N- és a Z-bit itt nem változik.

A többi utasítás leírásában általában nem fog szerepelni az, hogy milyen jelzőbitet állít, ezt az összefoglaló táblázatban (8.) lehet megnézni.

Az X- és Y-regiszter tartalmát az STX és STY utasításokkal lehet kiíratni tetszőleges tárhelyre, természetesen $0 és $FFFF határok között.

```
STX $0C01
STY $0C02
```

Az az Olvasó, aki olyan szerencsés helyzetben van, hogy hozzájut(ott) C 16-os vagy Plus/4-es géphez, máris próbálkozhat a gépi utasítások kipróbálásával. Célszerű ezt a könyvet a számítógéppel együtt használni; a legjobb, ha az utasításokat a megismerés után azonnal ki is próbáljuk. Ehhez bizonyos mértékig ismerni kell a gép beépített gépi kódú monitorának (TEDMON) a használatát. Ezt megtanulni a programozás tanulásával párhuzamosan célszerű. A könyvben viszont csak egy későbbi fejezetben van leírva, ezért az Olvasónak elkerülhetetlenül néha előre kell majd lapoznia.

A gépi kódú monitorba a MONITOR paranccsal lehet belépni (rövidítése: M (SHIFT)O). Ugyanezt a hatást lehet elérni azzal, ha a RUN/STOP billentyű folyamatos nyomvatartása mellett megnyomjuk a gép jobb oldalán lévő RESET gombot. Belépés után a gép visszajelzése:

```
MONITOR

          PC    SR  AC  XR  YR  SP
        ; 00FF  00  00  FF  00  F8
```

A felső sorban vannak jelölve a processzor egyes regiszterei, az alsóban a tartalmuk. A regiszterek kezdeti, tehát programfutás előtti értékét itt könnyen beállíthatjuk egyszerű felülírással. Ne felejtsük utána a RETURN billentyűt lenyomni!

Ezután következik a kipróbálandó utasítás, vagy utasítások beírása a gépbe. A BASIC-kel ellentétben itt majdnem tetszőleges, hogy melyik tárterületre kerül a programunk. (Ha nincs BASIC program a tárban, a $1000–$4000 terület mindegyik gépen szabad, ez egy kezdő számára mindenképpen elég nagy programterület.) Válasszuk pl. a $2000-es címtől kezdődő tartományt, s írjuk be:

```
A 2000 LDA #$01
```

A RETURN lenyomása után a kurzor a következő sorban, az alábbi kiírás után jelenik meg:

```
A 2002
```

Ide (utána) írjuk be, hogy BRK, és nyomjuk le kétszer a RETURN billentyűt. A két sorból álló programunkat ezzel be is fejeztük. (A BRK jelen esetben nagyjából azt a funkciót tölti be, amit a BASIC-ben a STOP vagy az END) Programunkat ki is listázhatjuk a

```
D 2000 2002
```

parancs segítségével. (BASIC-ben ez a LIST)

Eredménye:

```
.2000  A9 01  LDA #$01
.2002  00     BRK
```

A kiírt lista utasításait a BASIC-ben megszokott módon egyszerűen felülírhatjuk. A módosítás itt is a RETURN lenyomása után kerül a gépbe.

Futtassuk le a programunkat:

G 2000 (BASIC-ben ez a RUN vagy GOTO)

A futás végét a BREAK üzenet jelzi, és újra megjelenik a regiszterek tartalma. Ez azonban a futás utáni állapot! Mivel az LDA #$01-nek 01-re kellett állítania az akkumulátort, az AC alatti szám 01 kell, hogy legyen, bármi is volt az értéke a futás előtt.

A most leírt módszerrel majdnem mindegyik gépi utasítás hatása kipróbálható. Gépeljük be például az alábbi kis programtöredéket:

```
A 2000 LDA #$65
       STA $FF15
       BRK
```

(listázás: D 2000 2005 , futtatás: G 2000)

Az eddigiek alapján hatása a következő:

1. Az akkumulátor értéke $FF65 lesz.
2. A $FF15-ös tárhely felveszi az akkumulátor értékét, vagyis a $65-öt (alapérték: $F1). Mivel ezen a helyen a képernyőre vonatkozó információ van, a képernyő zöld színű lesz.

Fekete–fehér tv-t használók ehelyett a következő programot próbálják ki:

```
A 2000 LDA #$03
       STA $0FE7
       BRK
```

Hatására a képernyő jobb alsó sarkában egy C betű fog megjelenni. (A képernyő görgetésével természetesen feljebb kerülhet a többi szöveggel együtt.)

1.4.2. CÍMZÉSI MÓDOK

Amint eddig is megfigyelhető volt, az utasításszó (mnemonik) után túlnyomórészt egy olyan rész következik, ami arra utal, hogy a műveletet mivel kell elvégezni. Ez általában egy tárrekesz tartalma, amelynek a címét kell az utasítás után különféle módon megadni. Ezeket a megadási módokat nevezzük címzési módoknak.

Eddig két címzési módot ismertünk meg: az abszolút és a közvetlen (direkt) címzést.

Abszolút címzés

Az eddigiekben szereplő

```
LDA $FF19
LDX $FF15
STA $0C00
STY $0C02
```

utasításoknál az abszolút címzést használtuk. Ennél egyszerűen csak annak a tárhelynek a címét kell beírni, amire az utasítás vonatkozni fog. Az LDA $FF19 a $FF19-es tárrekesz tartalmát beolvassa az akkumulátorba, az STA $FF19 pedig az akkumulátor tartalmát bemásolja a $FF19-es tárhelyre.

```
BASIC megfelőik:
A = PEEK(65305)
X = PEEK(65301)
POKE 3072,A
POKE 3074,Y
```

Közvetlen vagy direkt címzés

Ez a címzés akkor szerepel, ha egy műveletet nem egy tárhely tartalmával, hanem egy előre megadott értékkel kell elvégezni.

Az eddigi példákban már szerepeltek ilyen címzések:

```
LDA #$01
LDX #$28
LDY #$08
```

Itt maga a $01-es, a $28-as és a $08-as szám töltődik az adott regiszterbe. Ezt a címzési módot mindig a szám előtti #-jel jelzi.

BASIC megfelelőik:

```
A = 1
X = 40 (dec.40 = hex.28)
Y = 8
```

Indexelt címzés

Ennél a címzési módnál az X- vagy az Y-regiszter segítségével valósul meg a címzés

```
LDA $0C00,X
```

Az utasítás által az akkumulátorba töltendő adat címét úgy kapjuk, hogy a megadott $0C00-ás címhez hozzáadjuk az X-regiszter tartalmát. Ha az X tartalma

éppen 0, akkor a fenti utasítás egyenértékű az LDA $0C00 utasítással, ha tartalma 5, akkor az LDA $0C05-nek felel meg.

Az X-regiszteren kívül, az Y-regiszterrel lehet még indexelt címzést megvalósítani

```
LDX $FD00,Y
LDA $1000,Y
```

Itt az Y-regiszter értéke adódik a címhez, és az így meghatározott tárrekesz tartalma töltődik be.

Az LDA $1000,X BASIC analógja:

```
A = PEEK(4096+X)
```

Indirekt indexelt címzés

Ez az egyik legbonyolultabb címzési mód. Itt a tárhely kijelölése többlépcsős.

Vegyük pl. az

```
LDA ($2B),Y
```

utasítást. Ez egy tárrekesz tartalmát az akkumulátorba tölti. Az, hogy melyik ez a tárrekesz, az

- a $2B és a $2C című rekeszek, valamint
- az Y-regiszter tartalmától függ.

Legyen pl. a

$002B tartalma = $01
$002C tartalma = $10

Ez a két rekesz együtt (fordított sorrendben leírva!) a $1001-es tárcímet jelöli ki. Ehhez az értékhez adódik még az Y-regiszter tartalma. Ha ez éppen 0, akkor a fenti utasítás a $1001-es rekesz tartalmát olvassa az akkumulátorba, de ha éppen 3 van benne, akkor az LDA $1004 utasítással egyenértékű a fenti utasítás.

Összefoglalva: az ebben a címzési módban szereplő és az azt követő rekeszek tartalma egy címet képez, melyet még az Y-regiszter módosíthat.

Figyeljünk arra, hogy a zárójelben levő cím értékének $00 és $FE közé kell esnie!

A címzési módban szereplő zárójelek itt arra utalnak, hogy nem a megadott címről kell betölteni, hanem a cím tartalma mutatja meg, hogy honnan. Ennél a címzési módnál kizárólag az Y-regiszterrel lehet indexelni. Sajnos indexelés nélküli indirekt címzés egyetlen kivételtől eltekintve nincs. (Ez a JMP indirekt.)

A fenti utasítás BASIC megfelelője:

```
A = PEEK(PEEK(43) + 256 * PEEK(44) + Y)
```

(A 43 a $2B decimális megfelelője)

Indexelt indirekt címzés

Bizonyos mértékig az indirekt indexelt címzéshez hasonló, itt is két tárhely mutatja az operandust.

Pl. nézzük a következő utasítást:

LDA ($2B,X)

Ha az X-regiszterben éppen 0 van, akkor a $2B és a $2C tárhely tartalma mutatja meg, hogy melyik tárrekeszt kell az akkumulátorba tölteni. Ha az X-regiszterben 2 van, akkor a $2B + 2 = $2D tartalmazza az indirekt címet. Tehát az X-regiszter értéke hozzáadódik a zárójelben megadott címhez, majd az így kapott tárcím és a rákövetkező cím tartalma (fordított sorrendben leírva) adja meg a betöltendő rekesz címét.

Legyen a

$2B tartalma 01
$2C tartalma 10
$2D tartalma 03
$2E tartalma 10

A fenti utasítással egyenértékű:

X = 0 esetén LDA $1001
X = 1 esetén LDA $0310
X = 2 esetén LDA $1003

Jelen esetben is igaz, hogy a zárójelbe csak $00 és $FE közé eső számot írhatunk.

Az utasítás BASIC megfelelője:

A = PEEK(PEEK(43 + X) + 256 * PEEK(44 + X))

A két utóbbi címzési mód elég körülményesnek látszik. Felmerülhet a kérdés, hogy a gyakorlatban hogyan és mire használják.

Az indirekt címzéseknél mindig két egymást követő tárcím tartalma (közösen, egybeolvasva) dönti el, hogy az utasítás mire vonatkozzon. Ez a két rekesz gyakorlatilag mutatóként működik amit "rá kell állítani" az éppen aktuális tárcímre (adatra), ezután a programunkban szereplő összes indirekt utasítás erre a címre fog vonatkozni. Az indexeléseket legtöbbször nem is szokták használni, ami itt azt jelenti, hogy 0-ra állítják az indexelő regisztert.

Nulláslap vagy 0. lapos címzés

Ez a címzési mód kevés újat jelent az eddig leírtakhoz képest. Kezdők számára nincs sok gyakorlati jelentősége.

Megértéséhez végezzünk el egy kísérletet, írjuk be:

A 2000 LDA $0034

A RETURN lenyomása után a képernyőn megjelenik:

A 2000 AD 34 00 LDA $0034
A 2003

A fenti utasítás tehát három rekeszt foglalt le:

1. $2000-es címen $AD, tehát az LDA abszolút cím kódja.
2.–3. $2001-es, $2002-es címen 34 00, tehát a $0034 alsó és felső byte-ja.
A 2003-as a következő szabad rekesz.
Próbálkozzunk újra:

A 2000 LDA $34

Visszajelzés:

A 2000 A5 34 LDA $34
A 2002

Ennek az utasításnak teljesen azonos a hatása, mégis csak két rekeszt foglal le:
1. $2000-es címen $A5, tehát az LDA 0. lapos abszolút című műveleti kódja.
2. $2001-es címen $34, a $0034 alsó byte-ja, a felsőt automatikusan 0-nak veszi.

A nulláslapos címzéssel nem csak helyet, de futási időt is megtakaríthatunk, ugyanis végrehajtása kevesebb időt igényel.

A legtöbb utasítást 0. lapos címzéssel is használhatjuk, sőt 0. lapos indexelt címzés is létezik.

A címzési mód elnevezése onnan adódik, hogy a tár első 256 rekeszét, tehát a $0000–$00FF címtartományt a tár 0. lapjának nevezik. Egy lap tehát 256 byte-nyi terület. A sorszámozás a 0-nál kezdődik. Hexadecimálisan leírva a felső byte mutatja meg egy címről, hogy hányadik lapra vonatkozik.

1.4.3. BELSŐ ÉRTÉKADÓ UTASÍTÁSOK

Az eddig megismert értékadásoknál az akkumulátor, az X- és Y-regiszterek egy-egy tárrekesz értékét vagy egy előre megadott értéket vettek át. Az utasításoknak a most bemutatásra kerülő csoportjában az egyes regiszterek között történik értékadás.

A TAX utasítás jelentése: az X-regiszter értéke legyen egyenlő az akkumulátor értékével, vagyis átmásoljuk az akkumulátor tartalmát az X-be.

Az akkumulátor tartalmának átmásolása az Y-ba a TAY utasítással lehetséges.

Az X értékét az akkumulátorba tölteni a TXA, Y-regiszter értékét pedig a TYA utasítással lehet.

Ezenkívül az X-regiszter és a veremmutató között lehet értéket átadni. A TSX a veremmutató tartalmát az X-regiszterbe tölti, a TXS pedig fordítva, az X tartalmát tölti a veremmutatóba.

Az akkumulátor, az X- és az Y-regiszter, valamint a veremmutató (stackpointer) között megvalósítható adatmozgás:

$$YR \leftrightarrow AC \leftrightarrow XR \leftrightarrow SP$$

Az itt leírt utasítások után nincs címzés, nem is lenne értelme.

Utasítás:	BASIC megfelelő
TAX	X = A
TAY	Y = A
TXA	A = X
TYA	A = Y
TSX	X = SP
TXS	SP = X

1.4.4. FELTÉTEL NÉLKÜLI UGRÁSOK

A legtöbb magasszintű programnyelvben találkozhatunk feltétel nélküli ugró utasítással. A BASIC-hez hasonlóan ez a GOTO utasítás szokott lenni. Hatása mindig az, hogy a programfutás nem a soronkövetkező utasítással, hanem az ugró utasításban megjelölt helyen folytatódik. Ilyen a JMP utasítás is.

```
JMP $3000
```

Hatására a JMP utáni utasítás helyett a $3000-es címen lévő utasításnál folytatódik a programfutás.

A JMP utasításnak van indirekt változata is.

```
JMP ($0312)
```

Hatása attól függ, hogy mi a tartalma a $0312-es és $0313-as tárhelynek. Bekapcsolás után:

– a $0312 -es tartalma $42

– a $0313 -as tartalma $CE

Az előbbi indirekt utasítás hatására a $CE42-re történik az ugrás (itt is fordított sorrendben értendő a cím).

Az indirekt ugró utasítás legközelebbi BASIC rokona az

ON M GOTO

típusú utasítás.

Amíg a programozói gyakorlatban az abszolút címzésű JMP utasítás sokkal gyakoribb, a gép rendszerprogramjaiban viszonylag sűrűn találkozunk az indirekt JMP utasítással. Ennek az az oka, hogy így a felhasználó könnyen módosíthatja, esetleg saját programjával helyettesítheti az operációs rendszer rutinjait. Mivel ezek a rutinok a ROM-ban vannak, felülírásukra nincs lehetőség, beavatkozni tehát csak az indirekt cím átírásával lehet.

Lássunk erre egy egyszerű példát.

A ROM $CE3F-es címén a JMP ($0312) utasítás van. Ez alapesetben a JMP $CE42-vel egyenértékű. $CE42-nél a STOP-billentyű vizsgálata, és a TI változó aktualizálása van. Sokszor kedvező, ha a BASIC programot nem lehet a STOP-pal megszakítani. Ezt úgy érhetjük el, hogy az indirekt címet $CE42-ről átírjuk $CE45-re.

Gépi programból:

```
LDA #$45
STA $0312
```

BASIC-ből: POKE 786,69

Ezután a JMP ($0312) utasítás $CE45-re ugrik, a STOP vizsgálatát, és a TI aktualizálását egyszerűen átugrottuk. Az alapállapot többféleképp is visszaállítható, például a RESET gomb megnyomásával.

1.4.5. SZÁMLÁLÓUTASÍTÁSOK

Aki írt már programot valamilyen magasszintű nyelven, az tudja, hogy milyen gyakori a ciklusszervezés (BASIC-ben például a FOR-NEXT). Gépi programoknál a ciklusokat leggyakrabban az X- és az Y-regiszterek segítségével szervezik. Az ezt támogató utasítások:

INX

Ezzel az utasítással az X-regiszter tartalmát 1-gyel növelhetjük.

Az Y tartalmát növelő utasítás az

INY

A regiszterek tartalmát 1-gyel csökkenteni a

DEX
DEY

utasításokkal lehet. Az akkumulátorra vonatkozó számlálóutasítás sajnos nincs.

Növelni és csökkenteni nem csak a regiszterek, hanem a tárrekeszek tartalmát is lehet. Erre szolgál az INC és DEC utasítás.

INC $1030

Ez a $1030-as rekesz tartalmát növeli 1-gyel.

DEC $2FFF

A $2FFF-es tartalmát csökkenti 1-gyel.

Az INC és DEC indexelt címzésű alkalmazásban:

INC $0832,X
DEC $0913,X

Fontos megjegyezni, hogy akár regiszterről, akár tárrekeszről van szó, tartalma mindenképpen $00 és $FF (255) közé kell, hogy essen. Ezért, ha $FF-et tartalmaz növelés előtt, akkor a növelés eredménye 0 lesz. Hasonlóképpen, ha 0 volt csökkentés előtt, akkor csökkentés után $FF (255) lesz.

A BASIC-analógokat a következő táblázat mutatja.

Utasítás:	BASIC
INX	X = X + 1
INY	Y = Y + 1
DEX	X = X - 1
DEY	Y = Y - 1
INC $0400	POKE 1024,PEEK(1024) + 1
DEC $0400	POKE 1024,PEEK(1024) - 1

Az eddig megismert utasításokból egy érdekes példaprogram állítható össze. Gépeljük be:

```
A 2000 LDA $00,X
       STA $0C00,X
       INX
       JMP $2000
```

Az első sor beolvas a $00–$FF tartomány valamelyik címéről az akkumulátorba. Ez a cím attól függ, hogy X értéke éppen mennyi. A második sorban a beolvasott érték a $0C00-ás cím utáni tárterületre kerül. Ez utóbbiról egyelőre elég annyit tudni, hogy a beírt értéktől függően a képernyő megfelelő karakterhelyén valamilyen karakter megjelenik. A harmadik sor növeli az X értékét, ezzel az első

és második sor működését is befolyásolja. A negyedik sor feltétlen ugrás az első sorra. Látható, hogy ez a program végtelen ciklusból áll, csak a RESET gombbal állítható meg. Természetesen nem szakítja meg az sem, ha az X-regiszter elérte a $FF értéket, mert ezután az INX hatására ismét 0 lesz és innen növekszik újra. Ilyen módon a programunk végigpásztázza a $00–$FF tartományt, és tartalmát "kivetíti" a képernyő tetejére.

Ezzel a "vetítőprogrammal" már el lehet kezdeni a gép felfedezését. (Biztosan több élményt nyújt, mint könyvekből kiolvasni.)

Az elindító parancs:

G 2000

A képernyő felső része azonnal tele lesz különféle karakterekkel. Feltűnő a bal szél közelében az óra, mely folyamatosan és viszonylag pontosan jár, miközben programunk fut. Középen egy jel akkor jelez, ha a STOP-gombot megnyomjuk, a jobb szélen egy másik minden billentyű lenyomására változik. Van olyan hely is, amelyik csak a SHIFT-, CTRL- (Control), és C=- (Commodore) billentyűre reagál.

Ezekre a magyarázat a könyv második fejezetében van.

1.4.6. LOGIKAI UTASÍTÁSOK

A logikai műveletek közül az ÉS a VAGY és a KIZÁRÓ–VAGY műveletek programozhatók. Ezek a műveletek annyiban hasonlítanak mondjuk a szorzáshoz vagy az összeadáshoz, hogy két érték között végezzük el, és az eredmény egy harmadik érték.

Vizsgáljuk meg egyenként ezeket a műveleteket:

Logikai ÉS (AND)

A logikai műveleteket mindig bitenként végzi a gép, tehát itt mindig kettes számrendszerben kell gondolkodni. (Ha máshogy nem megy, használjuk az átváltási táblázatot.)

AND # $71

utasítás a címzési módban megadott érték, – itt közvetlenül a $71-es szám – és az akkumulátor között végzi el a logikai ÉS műveletet.

Az AND műveleti szabályai:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

Tehát, az eredmény csak akkor 1, ha mindkét bit 1 értékű, egyébként 0.

Legyen az akkumulátor tartalma \$5F

\$5F	=	%01011111
\$71	=	%01110001
\$51	=	%01010001

Tehát a \$5F AND \$71 eredménye \$51.

A logikai művelet eredménye mindig az akkumulátorba kerül, fölülírva annak korábbi értékét.

Az AND műveletet jól lehet használni, ha egy tárhely bizonyos bitjeit úgy akarjuk 0-ra állítani, hogy a többi bit változatlanul maradjon.

Például az 1. hanggenerátor kikapcsolása a \$FF11-es tárhely 4. bitjének 0-ra állításával lehetséges. Az alábbi program ezt végzi el úgy, hogy a többi bitet változatlanul hagyja.

```
LDA $FF11
AND #$EF
STA $FF11
```

Az operandus olyan legyen, hogy 0 álljon azon a helyen, ahol 0-ra akarunk állítani egy bitet, és 1 legyen ott, ahol nem akarunk változtatni semmit. A fenti példában csak a 4. bitet akartuk 0-ra állítani, ezért az operandusnak (\$EF) csak a 4. bitje 0 értékű.

Logikai VAGY (OR) művelet

Erre a műveletre is igaz, hogy a címzési mód által meghatározott érték és az akkumulátor között bitenként végzi el a processzor. Az eredmény az akkumulátorban keletkezik.

Az OR művelet szabályai:

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

Az eredmény tehát csak akkor 0, ha mindkét bit 0, egyébként 1.

Nézzük meg, mit eredményez az

```
ORA # $71
```

utasítás, ha az akkumulátor tartalma \$93.

\$93	=	%10010011
\$71	=	%01110001
\$F3	=	%11110011

Az utasítás végrehajtása után az akkumulátorban $F3 lesz.

Az ORA műveletet akkor célszerű használni, ha bizonyos biteket úgy akarunk 1-re állítani, hogy a többi bit változatlan maradjon.

KIZÁRÓ VAGY művelet (EOR)

A VAGY művelethez hasonló, de az eredmény akkor is 0, ha mindkét megfelelő bit 1.

Műveleti szabályai:

0 EOR 0 = 0
0 EOR 1 = 1
1 EOR 0 = 1
1 EOR 1 = 0

Az eredmény 0, ha a bitek megegyeznek és 1, ha különböznek. Például az

EOR # $71

hatása, ha az akkumulátorban $93 van:

$93	=	%10010011
$71	=	%01110001
$E2	=	%11100010

A művelet elvégzése után az akkumulátorban $E2 lesz.

Az EOR utasítás alkalmas arra, hogy bizonyos bitek értékét ellenkezőjére (0-ról 1-re, 1-ről 0-ra) váltsunk anélkül, hogy a többi bit közben megváltozna. Erre jó példa a kisbetű/nagybetű - nagybetű/grafika üzemmód kapcsolása. (SHIFT C=).

Az alábbi programrészlet az aktuális üzemmódot az ellenkezőjére kapcsolja.

```
LDA $FF13
EOR #$04
STA $FF13
```

Az EOR #$04-gyel a 2. bitet (4-es helyi értékűt) az ellenkezőjére váltottuk, miközben a többi bit változatlan maradt.

A kisbetűs üzemmód kapcsolása:

```
LDA $FF13
ORA #$04
STA $FF13
```

Itt a 2. bitet 1-re állítottuk.

A nagybetűs üzemmód kapcsolása:

```
LDA $FF13
AND #$FB
STA $FF13
```

Ez a 2. bitet 0-ra állítja.

1.4.7. ARITMETIKAI UTASÍTÁSOK

Ehhez a témakörhöz az összeadás és kivonás utasításai tartoznak. Szorzóutasítás nincs, a szorzást csak elég bonyolult programmal lehet megvalósítani. Ilyen program van a BASIC ROM-ban, leírása a ROM-rutinokról szóló fejezetben szerepel.

Az aritmetikai műveleteknek is két operandusuk van, a műveletet az akkumulátor és a címzésben meghatározott tárhely (érték) között végzi a gép. Az eredmény az akkumulátorban keletkezik. Ebből az is következik, hogy az összeadandó számok közül az egyiket az akkumulátorba kell tölteni.

Pl., ha az akkumulátorban $17 van, akkor

```
ADC #$2A
```

összeadás eredménye: $17 + $2A = $41, decimálisan 23 + 42 = 65.

Abban az esetben, ha az összeadás eredménye nem fér el az akkumulátorban, az összeadás után a C jelzőbit (túlcsordulás vagy Carry) értéke 1 lesz, az eredménynek csak a 8 legalsó bitje kerül az akkumulátorba. Tehát az akkumulátorban az összeg helyett annál 256-tal ($100) kisebb szám lesz. Ha azonban a C-bitet úgy fogjuk fel, mint az akkumulátor 9. bitjét, akkor az eredmény most is pontos.

A C-bitnek ezenkívül más szerepe is van. Az összeadásnál ugyanis a címben meghatározott értéken kívül még a C-bit összeadás előtti értéke, 0 vagy 1 is hozzáadódik az akkumulátorhoz. A pontos számolás érdekében ezért minden összeadás előtt 0-ra kell állítani a C-t. Ezt a

```
CLC
```

utasítással lehet elvégezni.

Az ADC #$2A BASIC analógja az A = A + 42 + C

A következő programokban az összeadásra láthatunk példát.

```
CLC
LDA $3000
ADC $3100
STA $3200
```

A $3000-es címen levő számhoz hozzáadja a $3100-as tartalmát, és az eredményt a $3200-as helyen tárolja.

Ha nagyobb számokkal akarunk dolgozni, akkor a számot több tárhelyen kell tárolni. Számoljunk például 16 bites számokkal. Az első szám felső (azaz magasabb helyiértékű) 8 bitjét a $3000-es, az alsó 8 bitjét a $3001-es címen tároljuk. Ugyanígy egy másik számot a $3100–$3101-es címen. Az alábbi program ezt a két számot összeadja, és az eredményt a $3200–$3201-es címre teszi.

```
CLC
LDA $3001
ADC $3101
STA $3201
LDA $3000
ADC $3100
STA $3200
```

Az első összeadás előtt a C-bitet nullázni kell, de a második előtt már nem szabad, mert a "maradékot" tartalmazza, amit a felső összeghez még hozzá kell adni.

Az összeadáshoz nagyon hasonlóan valósul meg a kivonás, tulajdonképpen csak a különbségeket érdemes kihangsúlyozni. Itt nem fordulhat elő, hogy az eredmény $FF-nél nagyobb, viszont lehet 0-nál kisebb, ha egy kisebb számból vonunk ki egy nagyobbat. Ezt a C-bit 0 értéke jelzi. Ha a kivonás eredménye pozitív szám, vagy 0, akkor a C-bit értéke 1 lesz. A kivonás előtt a C-bit értékét 1-re kell állítani.

Ez a

```
SEC
```

utasítással lehetséges.

A kivonásra példa:

```
SBC #$2A
```

Ez az utasítás az akkumulátorból kivon $2A-t, valamint a C-bit értékének az ellentettjét.

BASIC megfelelője: A = A – 42 – (1 – C)

Az összeadásnál bemutatott példa megfelelője kivonásnál:

```
SEC
LDA $3000
SBC $3100
STA $3200
```

Ugyanez 16 bites számokkal

```
SEC
LDA $3001
SBC $3101
STA $3201
```

```
LDA $3000
SBC $3100
STA $3200
```

Két byte-nál hosszabb számok is kivonhatók és összeadhatók ugyanezzel a módszerrel. Csak a legelső kivonásnál kell a C-bitet állítani, utána már nem szabad. Figyelni kell arra is, hogy mindig a legalacsonyabb helyi érték felől kezdve kell a műveletet végezni.

Negatív számok, kettes komplemens

Térjünk vissza arra a speciális esetre, amikor egy kisebb számból vonunk ki egy nagyobbat. Például legyen az akkumulátor tartalma $17. Ekkor az

```
SBC #$2A
```

hatására az eredmény $ED (237) lesz, a C-bit jelzi az alulcsordulást. Ez az eredmény ugyanazon szabály szerint keletkezett, amely szerint a $FF (255) "után" a 0 következik, a 0 "alatt" pedig $FF van. A processzornak ez a tulajdonsága tette lehetővé a "hosszú" számok összeadását és kivonását, de a ciklusszervezésnél (INX) is kihasználtuk már. Erre épít ezen kívül a negatív számok kettes komplemens kódolása is.

A fenti kivonás eredménye $ED, ami a kettes komplemens kódolás szabályai szerint –$13 (–19) negatív érték. Nem véletlenül, hiszen

hexadecimálisan: 17–2A =–13
decimálisan: 23–42 =–19

Tehát, ha így fogjuk fel, akkor az SBC alulcsordulás esetén is jól számol. Bármely szám negatívját úgy kapjuk meg a kettes komplemensképzés szabálya szerint, hogy minden bitjét az ellenkezőjére váltjuk, majd ezután hozzáadunk 1-et.

Számoljuk ki, hogy 8 bites számot feltételezve mennyi a –5

00000101 (eredeti, +5)
11111010 (ellentett, $FA)

$FA + 1 = $FB   Tehát –5 = $FB (251)

Ha negatív számokkal is számolunk, akkor a legnagyobb helyi értékű bit jelzi az előjelet: 1=negatív, 0=pozitiv. A 8 bites számolásnál,

a pozitiv számok határai: 0-tól +127-ig,
a negativ számok határai: 0-tól –128-ig terjednek.

Ha vegyesen alkalmazunk pozitív és negatív számokat, akkor egyedül arra kell ügyelni, hogy ne legyenek a korlátok által megszabottnál nagyobbak ill. kisebbek. (Például 8 bites számnál +128 vagy ennél nagyobb.)

Hangsúlyozni kell, hogy a számolási eljárás ugyanaz az előjeles számoknál, mint a pozitív számoknál! Az összeadásnál és a kivonásnál leírt példák jól működnek akkor is, ha pozitív egészként értelmezzük a tárhelyek tartalmát, és akkor is, ha előjeles számként.

1.4.8. ÖSSZEHASONLÍTÁSOK, FELTÉTELES ELÁGAZÁSOK

Egyetlen programnyelv sem létezhet feltételes elágazás nélkül. A BASIC-ben ez például az

IF <feltétel> THEN <sorszám>

módon programozható. A gépi nyelvben ide két utasításcsoport tartozik:

- Összehasonlító utasítások;
- Feltételes ugró utasítások.

Egy átlagos feltételes elágazásnál először egy összehasonlító utasítás, utána egy feltételes elágazás van. Az összehasonlítás sok esetben elmaradhat.

Mivel egy összehasonlítás csak kétoperandusú lehet, itt is a címzésben megadott érték, és az akkumulátor között zajlik a művelet. Változás azonban csak a feltétel-regiszterben lesz. Az egyes jelzőbitek értéke úgy változik meg, mintha kivontuk volna a megadott tárhely tartalmát az akkumulátorból.

```
CMP #$2A
CMP $0428
```

A jelzőbitek a következőképp alakulnak:

Z = 1 : a két érték egyforma (az eredmény 0)
Z = 0 : a két érték különbözik (az eredmény nem 0)

N = 1 : a címzésben megadott érték nagyobb az
C = 0 akkumulátor tartalmánál. (alulcsordulás)

N = 0 : a címzésben megadott érték kisebb vagy egyenlő
C = 1 az akkumulátor tartalmánál.

A kisebb-nagyobb reláció eldöntéséhez az N- és a C-bitek bármelyike használható.

Fontos megjegyezni, hogy nem csak az összehasonlító utasítások állítják a jelzőbiteket, hanem majdnem mindegyik. Erről jó tájékoztatást ad az összefoglaló táblázat.

Amíg a CMP utasítás az akkumulátorral, a

```
CPX
CPY
```

utasítások az X- és az Y-regiszterrel végzik el az összehasonlítást. Egyébként hatásuk a CMP-vel megegyező.

Logikailag ide tartozik egy elég sajátos utasítás, a BIT utasítás is, mert ez is kizárólag a jelzőbiteket állítja.

BIT $07F8

Az utasítás hatására a $07F8 tartalmának a 7. bitjét az N jelzőbitbe tölti, a 6. bitjét pedig a V jelzőbitbe. Ezután AND műveletet hajt végre a tár és az akkumulátor között. Ha ennek eredménye $00, akkor a Z-bit értéke 1 lesz, ellenkező esetben 0.

Az N-, V-, Z-, C-bitek állapotától függő feltételes elágazásokat valósíthatunk meg a következő utasításokkal:

BMI $2009
BPL $2009

A BMI utasítás hatására a $2009-es címen folytatódik a program, ha az N-bit értéke 1, ellenkező esetben közvetlenül utána. A BPL utasításnál akkor történik ugrás, ha N = 0.

Az utasítások értelmezése:

CMP után BMI : ugrás, ha nagyobb
BPL : ugrás, ha kisebb vagy egyenlő

LDA után BMI : ugrás, ha a 7. bit=1. (MInusz)
BPL : ugrás, ha a 7. bit=0. (PLusz)

Pl. várakozás a STOP lemyomására:

```
. 2000 A5 91  LDA $91
. 2002 30 FC  BMI $2000
. 2004 00     BRK
```

A V bitet használó utasítások: BVS, BVC. A BVS-nél akkor történik ugrás, ha V = 1, a BVC-nél pedig akkor, ha V = 0. A V-bitet az ADC, SBC, BIT utasítások állítják. Leggyakrabban a BIT után fordul elő, de mindemellett ritkán használt utasítás.

A Z-bitet használó utasítások: BNE, BEQ

A BEQ utasításnál akkor történik ugrás, ha Z = 1, a BNE-nél pedig akkor, ha Z = 0.

Az utasítások értelmezése:

CMP után BEQ : ugrás, ha egyenlő
BNE : ugrás, ha nem egyenlő

```
LDA után BEQ : ugrás, ha 0
         BNE : ugrás, ha nem 0
```

Például várakozás egy billentyű lenyomására:

```
. 2000 A5 EF  LDA $EF
. 2002 F0 FC  BEQ $2000
. 2004 00     BRK
```

A C-bitet használó utasítások: BCS, BCC. A BCS utasításnál akkor történik ugrás, ha C = 1, a BCC-nél pedig akkor, ha C = 0.

Az utasítások értelmezése:

```
CMP után BCS : ugrás, ha kisebb vagy egyenlő
         BCC : ugrás, ha nagyobb

ADC után BCS : ugrás, ha túlcsordulás történt
         BCC : ugrás, ha nem történt túlcsordulás

SBC után BCS : ugrás, ha nem történt alulcsordulás
         BCC : ugrás, ha alulcsordulás történt
```

A feltételes ugróutasításoknál korlátozva van az ugrás "nagysága". Ha alacsonyabb címre ugrunk, akkor $7E (126) byte a maximális ugrási távolság, ha pedig nagyobb címre, akkor $81 (129) a legnagyobb ugrás az ugró utasítástól számolva. Mind a beépített monitor, mind az assembler programok hibajelzést adnak, ha ezt a szabályt meg akarjuk sérteni. Kezdő programozóknak talán nehéz, és nem is feltétlenül szükséges megérteni, hogy mi okozza ezt.

Amikor az ugró utasítás után 2 byte-os (hex. 4-jegyű) abszolút címet írunk az ugrás helyének megjelölésére, akkor ez egy kicsit félrevezető, ugyanis ez az utasítás a tárba kerülve már nem abszolút, hanem relatív címet használ az ugrás helyének megadására. Ez a relatív cím 1 byte-os kettes komplemens szerint értelmezett szám, ami tudvalevő, hogy +127-től -128-ig értelmezhető. A processzor úgy értelmezi a tárban az ugróutasítás utáni előjeles számot, hogy hozzáadja az éppen aktuális programszámláló értékhez, és ez adja az ugrás helyét.

Ne felejtsük azonban, hogy ha az ugrás nagyságára vonatkozó korlátozást betartjuk, akkor nem kell tudomást venni a relatív kódolási módról. Nyugodtan írjuk le mindig azt az abszolút címet, ahova ugorni kell.

Nézzünk egy olyan programozástechnikai fogást, amelyet az előbb leírt korlátozás miatt kell gyakran alkalmazni.

Legyen a feladat az, hogy beolvasunk a $0100-ás címről, ha ez nem 0, akkor a $D88B-nél kell a programnak folytatódnia, ellenkező esetben pedig a soronkövetkező címen. A probléma az, hogy BNE $D88B-t nem írhatunk, mert ez túl nagy ugrás lenne.

```
. 2000 AD 00 01 LDA $0100
. 2003 F0 03    BEQ $2008
. 2005 4C 8B D8 JMP $D88B
. 2008 8D 00 30 STA $3000
.
.
.               (folytatás)
```

A megoldás az, hogy csak abban az esetben engedjük "ráfutni" a programot a JMP-re, ha a feltétel teljesül, ellenkező esetben egyszerűen átugorjuk.

A listából a BEQ utasítás relatív kódolása is tanulmányozható. A $2003 tartalma $F0, ami a BEQ kódja, a $2004 tartalma pedig $03, ami +3 relatív ugrást jelent. Ugrás esetén ez a +3 a BEQ-t követő címhez ($2005-höz) hozzáadódik, így kapjuk meg az ugrás helyét, a $2008-at.

1.4.9. SZUBRUTINSZERVEZÉS

A BASIC nyelven programozók jól ismerik a szubrutinok (alprogramok) használatát. Ott a GOSUB utasítás hatására a vezérlés a szubrutinnak a GOSUB-ban megadott sorára kerül, majd amikor elér a RETURN utasításhoz, akkor visszatér a GOSUB utáni utasításra.

Gépi programoknál a szubrutinhívás a JSR utasítással történik. Csak abszolút címzésű változata létezik.

```
JSR $FFF0
```

A szubrutinokat RTS utasítással kell lezárni, hatására az utoljára hívó JSR utáni utasításon folytatódik a program. Értelemszerűen az RTS után semmiféle cím nem áll.

A szubrutinszervezéssel egyrészt áttekinthetőbb, másrészt rövidebb lesz a programunk. Ráadásul beépíthetünk programunkba olyan szubrutinokat, amelyeknek csak a funkcióját ismerjük.

```
LDA #$0D
JSR $FFD2
LDA #$41
JSR $FFD2
```

A $FFD2-nél levő ROM-rutinról egyelőre elég annyit tudni, hogy lényegében a PRINT utasítás funkcióját látja el. Meghívása előtt a kiírandó karakter ASCII kódját az akkumulátorba kell tölteni, majd ez a rutin kiírja a képrenyőre. Ez a program tehát először egy $0D (új sor) karaktert, majd egy $41-es karaktert, azaz egy A betűt ír ki.

Ez a példa is mutatja, hogy a gép ROM-jában viszonylag bonyolult funkciókra is van szubrutin. Rengeteg időt és energiát lehet használatukkal megtakarítani, nem is beszélve az olyan feladatokról, amelyet egy átlag programozó meg sem képes oldani.

A BASIC-hez hasonlóan itt is lehetőség van a szubrutinok egymásbaágyazására. Ez azt jelenti, hogy minden szubrutinban szerepelhet újabb szubrutinhívás.

A processzor a visszatérési címeket a veremben tárolja, és onnan keresi elő az RTS végrehajtásakor, ezért bizonyos veremmel kapcsolatos műveletek megzavarhatják az RTS normális végrehajtását. Erről a későbbiekben még lesz szó.

1.4.10. VEREMUTASÍTÁSOK

A veremtárat leginkább arra szokás használni, hogy egyes regiszterek értékét ideiglenesen tároljuk, majd amikor szükség van rájuk, ismét visszatöltsük. Az "ideiglenesen" itt azt jelenti, hogy ha valamit a verembe tettünk, azt előbb-utóbb ki is kell vennünk belőle, még akkor is, ha esetleg már semmi szükség sincs rá.

A veremutasítások:
PHA : az akkumulátor tartalma a verembe töltődik
PLA : a verem tartalma az akkumulátorba töltődik
PHP : az állapotregiszter tartalma a verembe íródik
PLP : a verem tartalma az állapotregiszterbe íródik

A verem tartalmán itt természetesen a verembe legutoljára beírt számot kell érteni, úgy is fogalmazhatunk, hogy a verem legfelső értékét.

Már volt szó róluk, de logikailag ide is tartoznak a TSX és TXS utasítások, melyek a veremmutató és az X-regiszter közötti értékátadást valósítják meg. Az alábbi programrész ezek használatára mutat egy példát.

```
TSX
INX
TXS
```

Ezzel a veremmutató értéke eggyel nő, mintha kivettük volna a legfelső értéket belőle. A következő PLA az utolsó előttinek beírt értéket fogja a veremben találni.

Nagyon fontos, hogy egy szubrutinon belül csak egyenlő számú verembe író, és veremből olvasó utasítást szabad végrehajtani. Ráadásul egy pillanatra sem haladhatja meg az olvasások száma az írásokét, de nem is lenne értelme. Ellenkező esetben az RTS végrehajtásakor rossz helyre térne vissza a vezérlés. A helyzet hasonló, mint BASIC-ben a zárójelek használatánál: nyitó zárójellel kell kezdeni, és minden zárójelet be kell zárni.

A verem legfelső elemét kiolvashatjuk anélkül, hogy eltűnne onnan:

```
TSX
LDA $0101,X
```

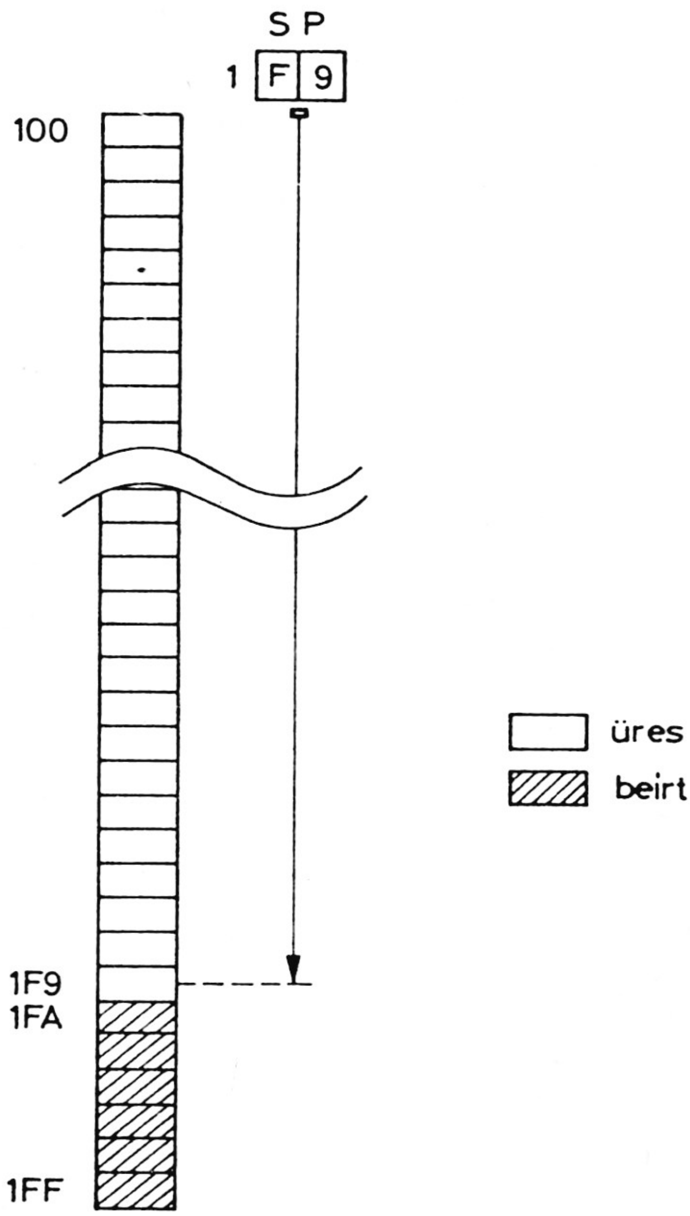
A PLA utasítással egyenértékű:

```
TSX
LDA $0101,X
INX
TXS
```

A PHA utasítással egyenértékű:

```
TSX
STA $0100,X
DEX
TXS
```

A következő ábra a veremtár felépítését mutatja be.



1.4.11. BINÁRIS ELTOLÁSI UTASÍTÁSOK

Ezek az utasítások egy bináris (kettes számrendszerbeli) szám bitenkénti jobbra vagy balra tolását végzik. Mivel egyoperandusú műveletről van szó, az eredmény abban a tárcímben keletkezik, amire a címzés vonatkozott.

ASL $3000

utasítás a $3000-es cím tartalmát bitenként balra tolja. A jobb szélen, a 0. bit helyére egy 0 lép be, a kicsorduló 7. bit pedig a C-bitbe kerül. A többi bit egy hellyel balra lép.

Legyen a $3000-es tartalma $65

$65 =%01100101 egy bittel balra tolva

$CA =%11001010, és a C-bitbe 0 íródott

Az utasítás végrehajtása után a $3000-es cím tartalma $CA lesz.

A ROL utasítás ciklikus balra léptetést valósít meg. Ez azt jelenti, hogy a C-bit értéke lép be a 0. bit megüresedett helyére, a kilépő 7. bit pedig a C-be kerül. Ily módon a processzor 9 bitet léptet "körbe".

$65 = %01100101, és legyen C = 1 (SEC)

az utasítás végrehajtása után:

$CB = %11001011, és C = 0

Az LSR utasítás az ASL-hez hasonló, de ellentétes irányú. A bitek most jobbra tolódnak, a megüresedett 7. bit értéke 0 lesz, a kilépő 0. bit pedig a C-bitbe kerül.

$65 = %01100101

az utasítás végrehajtása után:

$32 = %00110010, és C = 1

A ROR utasítás jelentése: ciklikus elforgatás jobbra. A C-bit értéke lép be a 7. bit megüresedett helyére, a kilépő 0. bité pedig a C-bitbe kerül.

$65 = %01100101, és legyen C = 1 (SEC)

az utasítás végrehajtása után:

$B2 = %10110010, és C = 1

Ezek az utasítások abszolút, és X-szel indexelt címzéssel is használhatók, de van olyan lehetőség is, hogy a műveletet az akkumulátorral végezzük el. A TEDMON monitort használók egyszerűen csak:

ASL
ROL
LSR
ROR

utasításokat írjanak, és az eltolás az akkumulátorra vonatkozik. A legtöbb assembler program, és néhány más monitor is

ASL A
ROL A
LSR A
ROR A

utasításokat vár ilyen esetben.

Az eltolások használata

Ezeket az utasításokat leggyakrabban a képernyőgrafikai programoknál és a szorzásnál (osztásnál) szokták használni. Amikor a bitek 0 ill. 1 értéke a képernyőpontok be- ill. kikapcsolt állapotát jelenti, akkor ezek mozgatása csak eltoló utasítással lehetséges. Egy kettes számrendszerbeli szám egy bittel balra léptetése a szám kettővel való szorzásával egyenértékű. Az ASL utasítás egyszerű kettővel való szorzás, ahol a túlcsordulást a C-bit jelzi. Az LSR kettővel való osztást jelent, a C-bitbe a maradék kerül.

Legyen például a $3000–$3001-es címen egy 16 bites szám, (az alsó byte a $3000, a felső a $3001). Szorozzuk meg kettővel, az eredményt tegyük a $3100–$3101-es címre.

```
LDA $3000
ASL
STA $3100
LDA $3001
ROL
STA $3101
```

A kilépő 7. bit a $3101-es cím 0. bitjére, azaz a hosszú szám 8. bitjére került. Ezzel a módszerrel lehet egy hosszabb tártartományt összefüggően, bitenként eltolni.

Ha azt akarjuk, hogy az eredmény ott keletkezzen, ahol az eredeti szám volt ($3000–$3001) :

```
ASL $3000
ROL $3001
```

Ugyanez, de 4-gyel szorozva:

```
ASL $3000
ROL $3001
ASL $3000
ROL $3001
```

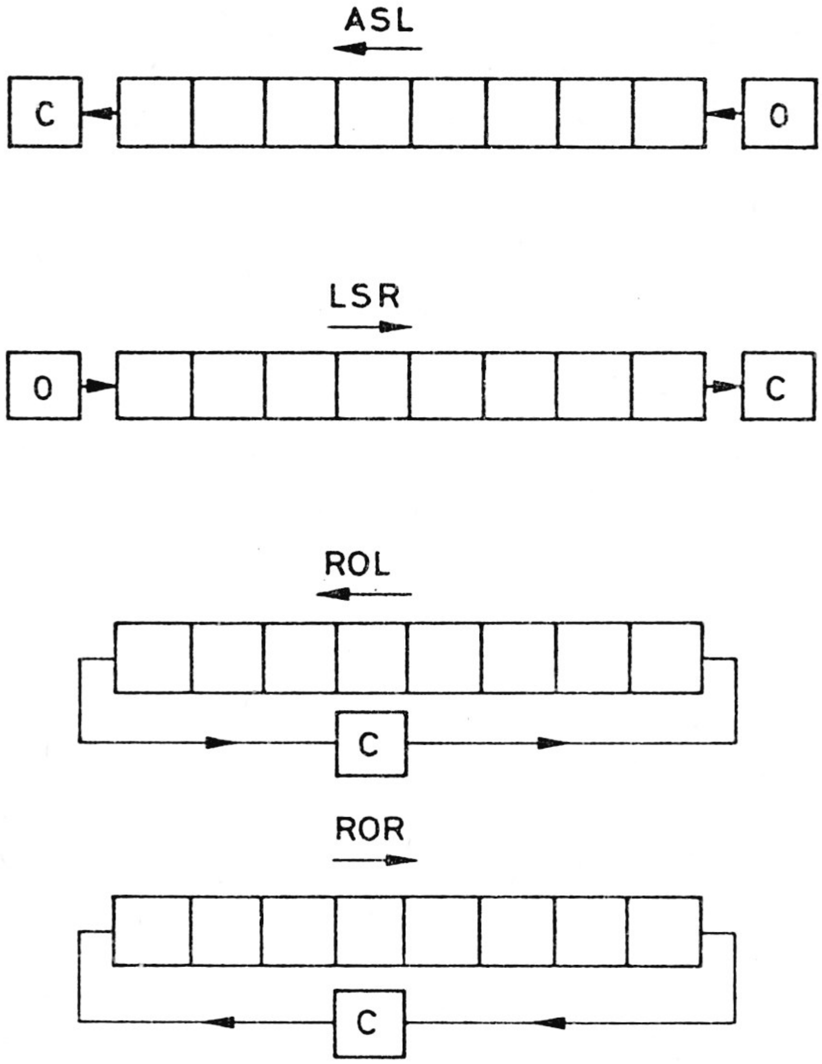
Vagyis kétszer kell kettővel szorozni.

Ugyanez, de 2-vel osztva:

```
LSR $3001
ROL $3000
```

Az osztási maradék (0 vagy 1) a C-bitbe kerül.

Az ábrán az eltolóutasítások modelljét mutatjuk be.



1.4.12. AZ ÁLLAPOTREGISZTER UTASÍTÁSAI

Ezek az utasítások az állapotregiszter bitjeit állítják 0-ra vagy 1-re. Ezek közül kettőt már korábban ismertettünk:

- a SEC utasítás a C-bit értékét 1-re állítja. A kivonás (SBC) előtt kötelező.
- a CLC utasítás a C-bit értékét 0-ra állítja. Ez pedig az összeadás (ADC) előtt kell.

Az I-kapcsoló állításával engedélyezhetjük vagy letilthatjuk a megszakítást. A SEI utasítással I értékét 1-re állítjuk, a megszakítást ezzel letiltottuk. A CLI utasítás 0-ra állítja az I-kapcsolót, ezzel engedélyezzük a megszakítást. A megszakítás témaköréről szól a 2.4.-es alfejezet. Addig csak annyit kell tudni erről, hogy normális működés közben $I = 0$.

A V-bitet 0-ra állíthatjuk a CLV utasítással. Az 1-re állításra közvetlen utasítás nincs.

A D-kapcsoló 1-re állítása a SED, 0-ra állítása a CLD utasítással történik. Ezzel állítható a bináris és a decimális üzemmód.

A processzor kétféle üzemmódban tud dolgozni. A bináris üzemmódban kettes számrendszerbeli számokkal, decimális üzemmódban BCD számokkal, vagyis binárisan kódolt decimális számokkal. Az eddig leírtak mind a bináris üzemmódra vonatkoztak. A decimális mód használata gyakorlatilag elenyésző, de azért ismerkedjünk meg ezzel is. Decimális üzemmódban a 8 bit két négybites részre oszlik. Ez a két négybites rész egy-egy tizes számrendszerbeli számjegyet tartalmaz. Így az akkumulátor tartalmának 1 és 99 közé kell esnie. Arra kell csak ügyelni, hogy ne használjunk olyan számokat, melyek hexadecimálisan leírva A, B, C, D, E, F számjegyeket tartalmaznak. Példaprogramunk

```
SED
LDA #$09
CLC
ADC #$01
CLD
```

lefuttatása után az akkumulátorban $10 lesz, tehát $09 + $01 = $10 módon számolt a gép. Ezt most nem $10-ként (16), hanem decimális 10-ként kell értelmezni. Ugyanez a program bináris módban $0A-t eredményez.

Bekapcsolás után a gép azonnal bináris üzemmódra kapcsol, és soha nem vált át decimálisra. A decimális mód használhatóságát csökkenti, hogy túlcsordulás esetén a Z-bit nem működik.

1.4.13. EGYÉB UTASÍTÁSOK

A most leírásra kerülő utasítások semmilyen más kategóriában nem fértek el.

A NOP utasításnak semmilyen hatása sincs, annak ellenére, hogy egy tárhelyet lefoglal és "végrehajtásához" kell bizonyos idő. A monitorral programozók úgy szokták használni, hogy programjuk egyes helyeire NOP-okat tesznek, fenntartva a helyet egy esetleges későbbi bővítésnek. Három egymásutáni általában elég, mert három helyet foglal el egy JSR.

A BRK utasítást nyugodtan használhatjuk anélkül, hogy működési mechanizmusában elmélyednénk. Amikor erre kerül a vezérlés, a programfutás megszakad, a monitorba térünk vissza, és a képernyőn megjelenik a regiszterek tartalma. Kiválóan tesztelhetők segítségével az egyes programrészletek, sőt akár egyetlen utasítás hatása is.

A BRK tulajdonképpen megszakítás utasítás és rokon a JSR utasítással is. Hatására itt is a verembe kerül az utasításszámláló aktuális értéke (visszatérési cím), de itt még a feltételregiszter tartalma is. Utána végrehajtásra kerül az a rutin, melyre a $FFFE–$FFFF címek tartalma mutat. Ebből az RTI hatására tér(ne) vissza. A C 16-os és a Plus/4-es gépeknél azonban a BRK-rutin a fent leírt hatást eredményezi, a programfutás tehát megszakad. Ezekről a 2.4.-es alfejezetben lesz még szó.

1.5. A TEDMON monitor használata

A C 16-os és a Plus/4-es gépek rendelkeznek egy beépített monitorral, ami egy disassemblert, valamint egy gyengébb assemblert is tartalmaz. Ezzel a monitorral közvetlenül a tárba írhatjuk a gépi kódú programunkat, és futtathatjuk azt. A következőkben a TEDMON monitor használatát, parancsait és lehetőségeit tárgyaljuk.

A TEDMON monitor utasításkészlete

A – Egy assembly sort lefordít gépi kódra (ASSEMBLE)

C – A tár két területét összehasonlítja (COMPARE)

D – A tár adott területét lefordítja assembly nyelvre (DISASSEMBLE)

F – Egy megadott értékkel feltölt egy területet (FILL)

G – A meghatározott címnél elkezdi a futtatást (GO)

H – A tár egy megadott részét végignézi, és kikeres bizonyos byte-sorozatokat (HUNT)

L – Betölt egy file-t lemezről vagy kazettáról (LOAD)

M – A tár adott területének a tartalmát megjeleníti hexadecimálisan és karakteresen (MEMORY)

R – Kiírja a 8501-es regisztereinek a tartalmát (REGISTERS)

S – A tár adott területének kiírása egy file-ba kazettára vagy lemezre (SAVE)

T – Egy megadott tárterület átmásolása máshova (TRANSFER)

V – A lemezre vagy kazettára rögzített file összehasonlítása a tártartalommal (VERIFY)

X – Kilépés a TEDMON-ból (EXIT)

. – Egy sort fordít gépi kódra (= A)

⟩ – Memória-módosítás

; – A regiszterek tartalmának módosítása

A TEDMON-ba a MONITOR paranccsal, vagy ennek a rövidítésével, az M(SHIFT)O paranccsal léphetünk be. Ezután a monitor a regiszterek tartalmának kiírásával várja az utasításokat.

FIGYELEM: a monitorban csak a tizenhatos számrendszerbeli számok használhatók.

A következőkben az utasításokat és azok hatásait ismertetjük.

ASSEMBLE

UTASÍTÁS: **A** <cím> <mnemonik> <címzési mód>

Egy assembly sor gépi kódjának a tárba fordítása. A RETURN billentyű lenyomása jelzi a sor végét. Hatására a TEDMON lefordítja a sort gépi kódra és elhelyezi a tárban, majd az A betűt és a következő címet kiírva várja az új sort. Ez teszi lehetővé, hogy nekünk csak a programírás kezdetén kell beírni az A betűt és a kezdőcímet, a továbbiakban a TEDMON kiszámolja a következő utasítás helyét. Ha a sorban hiba van, akkor azt a sor végén megjelenő kérdőjel jelzi. A TEDMON nem jelez hibát, ha ROM-ba kísérelünk meg írni, de ekkor a beírt utasításunk ellenére az az utasítás fog megjelenni a képernyőn, ami a ROM címen eredetileg van.

<cím> – kijelöli, hogy az utasítás kódja hova kerüljön a tárban.

<mnemonik> – a 8501-es processzor valamely utasítása assembly nyelven.

<címzési mód> – az utasításhoz kapcsolható lehetséges címzési módok egyike.

Nehézséget jelent, hogy itt nem használható semmilyen cimke, így az előre történő ugrást nem tudjuk kiszámítani. Egyetlen könnyebbség, hogy feltételes elágazáskor az abszolút címet kell beírni, a relatív címet a TEDMON kiszámolja és lefordítja.

```
A 2000 LDA #$00
```

A RETURN lenyomására az LDA kódja, valamint a 00 érték bekerül a tárba, majd a TEDMON kiszámítja és kiírja a soron következő utasítás kezdőcímét:

```
A 2000 A9 00 LDA #$00
A 2002
```

Megjegyzés: a pont (.) azonos az A utasítással.

COMPARE

UTASÍTÁS: **C** <1. cím> <2. cím> <3. cím>

Az <1. cím>-től a <2. cím>-ig terjedő tárterület összehasonlítása a <3. cím>-től kezdődő, azonos hosszúságú tárrésszel. Ha a két tárterület megegyezik, akkor a monitor nem jelez ki semmit. Eltérések esetén az eltérések helyét írja ki.

<1. cím> – az első tárterület kezdőcíme (ezt már hasonlítja)

<2. cím> – az első tárterület végcíme (ezt még hasonlítja)

<3. cím> – a második tárterület kezdőcíme.

Pl. C 2000 2100 3000

DISASSEMBLE

UTASÍTÁS: **D** <1. cím> <2. cím>

A tár <1. cím>-től <2. cím>-ig tartó részét disassemblálja, azaz az ott található tárcímek tartalmát gépi kódú programnak tekintve kiírja azok címét, hexadecimális tartalmát, mnemonikját és az operandust. Ha nem jelöljük meg a disassemblálandó terület végcímét a <2. cím>-mel, akkor a disassembler a következő 20 byte-ot disassemblálja. Ha kezdőcímet sem adunk meg, akkor az utoljára olvasott (vagy írt) helytől kezdődik a disassemblálás. Ha egy értékhez nem tud mnemonikot kapcsolni (illegális érték), akkor a helyére ??? kérdőjeleket ír. A disassemblált listában lehetőség van a javításra. Egyszerűen a kurzorral a javítandó részre lépünk, majd a javítás után a RETURN lenyomására a TEDMON az A parancs végrehajtásával kijavítja a sort.

<1. cím> – a disassemblálandó terület kezdőcíme (nem kötelező megadni)

<2. cím> – a disassemblálandó terület végcíme (nem kötelező megadni)

D F427 F435

```
F427  90  06      BCC  $F42F
F429  AE  33  05  LDX  $0533
F42C  AC  34  05  LDY  $0534
F42F  8E  33  05  STX  $0533
F432  8C  43  05  STY  $0534
F435  60          RTS
```

FILL

UTASÍTÁS: **F** <1. cím> <2. cím> <érték>

Az <1. cím>-től a <2. cím>-ig terjedő tárterületet a megadott <érték>-kel tölti fel. Ez az utasítás egyes területek azonos értékkel való feltöltését (pl. kinullázását) eredményezi. A tár tesztelésekor, egyes utasítások hatásának vizsgálatakor hasznos.

Ha a feltöltés után az adott címről nem ugyanaz az érték olvasható vissza, mint amit beírattunk, kijelzésre kerül a cím.

<1. cím> – az első feltöltendő tárhely címe

<2. cím> – az utolsó, még feltöltendő tárhely címe

<érték> – az az érték, amellyel az adott területet fel kívánjuk tölteni.

F 2000 2100 00

GO

UTASÍTÁS: **G** <cím>

A <cím>-ben megadott értéktől kezdi végrehajtani az ott lévő gépi kódú programot. Az utasítás végrehajtásakor a <cím> értéke betöltődik a programszámlálóba, majd a végrehajtás az új címtől kezdődik. Ha a <cím> értéket elhagyjuk az utasítás után, akkor a végrehajtás a PC-ben kijelzett értéktől kezdődik. A BRK utasításra a futás megszakad, és visszatér a monitorba.

A $8000–$FFFF tartományban ez az utasítás a rendszer ROM-ra vonatkozik.

HUNT

UTASÍTÁS: **H** <1. cím> <2. cím> <adatok>

Az <1. cím>-től a <2. cím>-ig terjedő tárterületen megkeresi az <adat>-ban szereplő byte-sorozatot és kiírja az előfordulási helyek kezdőcímét, ha nem talál ilyet, akkor nem jelez ki semmit.

<1. cím> – a vizsgálandó tárterület kezdőcíme

<2. cím> – a vizsgálandó tárterület végcíme

<adatok> – a keresett byte-sorozat

Az adat lehet numerikus vagy füzér (string) formátumú. Ha numerikus adatról van szó, akkor azt hexadecimális alakban kell megadni. Több adat (byte-sorozat) esetén azokat szóközzel kell elválasztani.

H 8000 8080 A9 4C

Ez a parancs az LDA #$4C utasítást keresi meg.
Ha az adat füzérkifejezés, akkor azt felső vesszővel (aposztroffal) kell bevezetni.

H 8000 FFFF 'READY

Az utasítás a rendszerterületen megkeresi és kiírja a READY felirat kezdőcímét.

LOAD

UTASÍTÁS: **L** "file-név",<egységszám>

Egy file betöltése kazettáról vagy lemezről. Az L parancs hatására a TEDMON betölti a "file-név" alatt szereplő file-t a tárba. A file arra a területre kerül vissza, ahonnan kiírták.

"file-név" – a betöltendő file neve (idézőjelbe kell tenni)

<egységszám> – a géphez csatlakoztatott és használni kívánt tárolóegység száma (a kazettáé 1, a lemezmeghajtóé 8)

L "PROG",1 – kazettáról való betöltés
L "PROG",8 – lemezről való betöltés

Megjegyzés: az egységszám előtti vessző helyett szóköz is használható. Az alapértelmezés a kazetta, ha nem adunk meg egységszámot, akkor az utasítás erre vonatkozik.

MEMORY

UTASÍTÁS: **M** <1. cím> <2. cím>

A tár kijelölt részét a képernyőre írja hexadecimális és karakteres formában. A tártartalom módosítása közvetlenül a képernyőn elvégezhető. Az M utasítással

ki kell listázni a képernyőre a módosítandó tárterületet, majd a kurzorral rá kell menni a javítandó értékre és beírni az új értéket. A RETURN lenyomása után a tárhely már a módosított értéket tartalmazza. A javítás az inverz karakteres részen nem lehetséges. Ha folyamatosan szeretnénk egy nagyobb területet vizsgálni, akkor az első alkalommal csak a kezdőcímet adjuk meg. A továbbiakban az M és a RETURN billentyűkkel lapozhatunk tovább. Ha a tártartalom nem jeleníthető meg karakteresen, akkor a helyén pont (.) jelenik meg.

<1. cím> – a megjelenítendő tárrész kezdőcíme (nem kötelező megadni, elhagyása esetén a megjelenítés az utoljára olvasott/írt értéktől kezdődik)

<2. cím> – a megjelenítendő tárterület végcíme (szintén elhagyható, ha nem adjuk meg, akkor 12 sornyi tárrész (96 byte) jelenik meg)

A listázás formátuma a következő:

M 8000 8018

cím	8 byte tartalma numerikusan							karakteresen	
>8000	4C	19	80	4C	0A	80	00	43	:L.. L... C
>8008	42	4D	20	CC	FF	20	D8	8A	:BM L. X.
>8010	85	13	20	C9	C7	58	4C	7E	:.. IGXL .
>8018	86	20	17	81	20	2E	80	20	:. .. ..

UTASÍTÁS: > <cím> <1. adat> ... <8. adat>

1–8 db tárhely közvetlen módosítása. A módosítás után megjelenik az az érték, ami a beírás után az adott címről éppen kiolvasható volt.

<cím> – a módosítás kezdőcíme hexadecimális alakban (erre a helyre kerül az <1. adat>)

<adatok> – a <cím>-től kezdődően ezekre az értékekre módosul a tár tartalma (az <1. adat> a <cím>-ben megadott helyre kerül, ha nem írunk adatokat, akkor csak a jelenlegi tartalom kijelzése történik meg)

>2000 EA – a 2000 helyre EA érték kerül

>2000 EA A9 00 60 a 2000-ra EA
a 2001-re A9
a 2002-re 00
a 2003-ra 60 kerül.

REGISTERS

UTASÍTÁS: **R**

A képernyőre írja a 8501-es processzor regisztereit, és azok tartalmát.

R

PC SR AC XR XR SP
; 1005 00 00 00 01 F9

Megjegyzés: a pontosvessző (;) ugyanúgy használható a regiszterek módosítására, mint a > jel a tárcímek közvetlen átírására.

SAVE

UTASÍTÁS: **S** "file-név",<eszközszám>,<1. cím>,<2. cím>

A RAM <1. cím>-től <2. cím>-ig terjedő részének kiíratása az <eszközszám> által meghatározott tárolóegységre.

"file-név" – a tárolt adatok az itt megadott nevű file-ban lesznek a tárolóegységen

<eszközszám> – annak a tárolóegységnek a száma, amire rögzíteni akarjuk a tárrészt (a kazettás egység 1, a lemezegység 8)

<1. cím> – a tárolandó terület kezdőcíme (e cím tartalmát már tárolja)

<2. cím> – a tárolandó terület végcíme (E cím tartalmát már nem tárolja. Ha ismerjük az utolsó, még tárolni kívánt adatbyte címét, akkor a <2. cím>-be ennél eggyel nagyobbat kell írni.)

S "PRÓBA",1,1001,4000

Ez a példa PRÓBA néven kazettára rögzíti a C16-os teljes szabad tárterületét.

Megjegyzés: Az S paranccsal rögzített file programfileként kerül kiírásra. Későbbi visszatöltése során (L parancs) ugyanoda kerül vissza, ahonnan kimentették.

TRANSFER

UTASÍTÁS: **T** <1. cím> <2. cím> <3. cím>

<1. cím>-től <2. cím>-g terjedő tárterület áttöltése a <3. cím>-től kezdődő új területre. Az áttöltés történhet lefelé és felfelé egyaránt, azonban a <3. cím> értéke nem eshet az <1. cím> és a <2. cím> értékei közé.

<1. cím> – az áttöltendő terület kezdőcíme (ezt a tárcímet már átmásolja)

<2. cím> – az áttöltendő terület végcíme (ezt a tárcímet még átmásolja)

<3. cím> – az új terület kezdőcíme (ide kerül az <1. cím>-ben megadott tárcím tartalma)

T D000 D7FF 3800

Példánk a C16-os és Plus/4-es eredeti karakter ROM-ját másolja át RAM területre $3800-tól kezdődően.

VERIFY

UTASÍTÁS: **V** "file-név",<eszközszám>

A tárolóegységen lévő file-ot összehasonlítja a tár azonos kezdőcímű és hosszúsá-

gú részével. Először megnézi a tárolt file kezdőcímében, hogy a tár melyik címétől kezdve vitték ki az adatokat. Az összehasonlítás ettől a címtől kezdődik. Hiba esetén a VERIFY ERROR hibaüzenet jelenik meg, egyezéskor nem ír ki semmit.

"file-név" – az összehasonlítandó file neve

<eszközszám> – az előzőekben már említett tárolóegységek száma (1 = kazetta, 8 = lemez)

V "PRÓBA",8

EXIT
UTASÍTÁS: X

Visszatérés a BASIC-be. Az esetlegesen már előzőleg a tárban lévő BASIC programunk újra listázható és futtatható, persze csak abban az esetben, ha a monitor valamelyik parancsával nem tettük tönkre a programterületet.

1.6. Programozás assemblerrel

A számítógép gépi kódú programozása során az egyes elemi utasításoknak megfelelő bitsorozatot kell a tárba juttatnunk. Ehhez fejből kellene tudnunk az összes gépi utasítás kódját, ha a gépi programozást könnyítő programok nem állnának a rendelkezésünkre. Az első számítógépeket még csak így, számokkal lehetett programozni. A következő lépcsőfok az ASSEMBLY nyelv megalkotása volt.

Ez még tulajdonképpen gépi kódú programozás, annyiban azonban könnyít a dolgon, hogy minden elemi utasításnak, – összefüggésben a funkciójával – külön nevet adtak. Ezeket a neveket sokkal könnyebb megtanulni, mint az utasításkódokat. Az assembly nyelven megírt programot, amit forrásnyelvű programnak nevezünk, az ASSEMBLER program segítségével lehet gépi kódra fordítani. Az assembler program soronként végigmegy a forrásnyelvi programon, és az ott álló gépi utasítások neveit, a mnemonikokat helyettesíti a hozzájuk tartozó gépi kóddal, majd elhelyezi a tár adott részén. Tulajdonképpen ez egy fordítás, assembly nyelvről gépi kódra, tehát tekinthetjük az assembler programot egy speciális fordítónak is.

A C 16-os és a Plus/4-es gépek tartalmaznak a monitorban egy beépített assembler programot. Ez azonban nem igazán assembler, mert a begépelt sorokat azonnal a tárba fordítja, így nincs lehetőségünk a program összeszerkesztésére. Minden előforduló feltételes és feltétel nélküli ugrási címet már előre ki kell számolnunk, valamint esetleg előre be kell írni olyan címeket is, amelyek értéke a programírás során csak később fog kiderülni.

```
3000 LDX  #$00
3002 TXA
3003 STA  $0C00,X
3006 INX
```

```
3007 BEQ $300C
3009 JMP $3003
300C BRK
```

A BEQ beírásakor az utána írandó $300C értéket még nem ismerjük, hiszen nem tudjuk, melyik tárcímre kerül a majd BRK utasítás. Ezt a címet vagy előre kiszámoljuk (ami ilyen közeli ugrási cím esetén még egyszerű, de egy távolabbi címet már nehezen tudnánk pontosan meghatározni), vagy pedig beírunk egy tetszőleges értéket, és ha már megkaptuk a pontos címet, akkor visszamegyünk és kijavítjuk a helyes értékre.

Ez még csak az első nehézség. Az igazi probléma a hibajavítás során merül fel. Törölni még lehet egy sort vagy rövidebb részt, de komolyabb változtatás már hosszadalmas, aprólékos munka és sokszor a program teljes tönkretételét eredményezi. Ha pl. egy kétbyte-os utasítást ki akarunk törölni, akkor beírhatunk két NOP-ot a helyére. Ez a megoldás azonban hosszabb programrészek megváltoztatásánál nem célravezető, mert növeli a tárigényt és lassítja a program futását. Új utasítás vagy utasítások beszúrása azonban még nehezebb, hiszen előbb helyet kell biztosítani számukra (ami esetenként igen bonyolult feladat), és csak azután írhatjuk le a kívánt utasításokat.

A MONITOR-ban lévő A paranccsal, tehát közvetlenül a tárba írjuk be az utasításokat és már a tárban lévő kész gépi kódú programot tudjuk javítani, alakítani. Egy igazi assembler programnál a programot a BASIC-hez hasonló módon állítjuk össze, és csak ezután kerül a kész program lefordításra. Maga az assembler program tehát csak a fordítást végzi és a kódokat írja be a tárba.

A forrásnyelvi program összeállítására a gép eredeti BASIC szövegszerkesztőjét használhatjuk. A programírás nagyon hasonlít egy BASIC program megírásához. Itt is sorszámmal kell kezdeni egy sort, majd ezután írhatjuk az utasítás nevét és operandusát. A sorszámokra azért van szükség, hogy két sor közé könnyen új sort szúrhassunk be. A monitor programmal ellentétben itt új sor beszúrása soha semmilyen nehézséget nem jelent.

Az assemblereknek nagyon sok olyan szolgáltatása van, ami a programozó dolgát jelentősen megkönnyíti. Tetszés szerint használhatunk pl. bináris, decimális és hexadecimális számokat. A legnagyobb könnyebbség azonban az, hogy az abszolút tárcímek helyett cimkéket (szimbólumokat) használhatunk. Az assembler megengedi, hogy cimkékkel megjelöljünk egyes sorokat (nevet adjunk nekik), így nem kell (előre) tudnunk egy utasítás kezdőcímét, mert egyszerűen a nevével hivatkozhatunk rá. Ezeket a szimbólumokat célszerű szubrutinok belépési pontjaira tenni, vagy olyan sorok elé, amelyekre ugró vagy egyéb utasítás hivatkozik.

Az assembler a fordítás során kiszámolja, hogy az egyes cimkéknek milyen abszolút cím felel meg és behelyettesíti ezt az értéket minden olyan helyre, ahol hivatkoztunk rá.

A fejezet elején említett példaprogramunk a következőképpen néz ki assembler forrásnyelven:

```
10        *=   $3000
20 KEZD   LDX  #$00
30        TXA
40 KIIR   STA  $0C00,X
50        INX
60        BEQ  VEGE
70        JMP  KIIR
80 VEGE   BRK
```

Ha ezt a programot lefordítjuk egy assembler programmal, akkor teljesen ugyanazt a programot kapjuk, amit monitorral közvetlenül a tárba gépeltünk. A 60-as sor írásakor azonban elég csak azt tudni, hogy majd később lesz egy programsorunk, amit a VEGE cimkével fogunk jelölni.

Az assembler fordításkor végigmegy a mi forrásnyelvű programunkon, és kiszámolja a szimbólumoknak megfelelő abszolút címeket:

```
KEZD = $3000
KIIR = $3003
VEGE = $300C
```

Ezután behelyettesíti a BEQ és JMP után az igazi értékeket, és ebben a formában írja a tárba a kész programot.

Ha egy új utasítást akarunk beszúrni az STA és az INX közé, ez assemblernél egyszerű; be kell írni egy új sort.

```
45 STA $0800,X
```

Monitornál ez sokkal bonyolultabb; a program második részét újra be kell gépelnünk.

Az első sorban a "*=" kifejezéssel a program kezdőcímét adtuk meg, ettől a címtől kezdődően kerül a program a tárba.

Cimkékkel nemcsak utasításokat, hanem közvetlenül tárcímeket is elnevezhetünk.

```
 10 TABL1 =     $1000
 20 TABL2 =     $2000
 30       *=    $3000
 40 KEZD  LDX   #$00
 50 KIIR  LDA   TABL1,X
 60       STA   TABL2,X
 70       INX
 80       BEQ   VEGE
 90       JMP   KIIR
100 VEGE  BRK
```

Ez a kis program a $1000-es helyen kezdődő, TABL1 nevű táblázatot másolja át $2000-től kezdve a TABL2 nevű táblázatba. Itt tehát a cimkék nem csak

utasításokat vagy belépési pontokat jelöltek, hanem egy táblázat kezdetét is, amire később hivatkozni lehet (LDA TABL1,X ; STA TABL2,X).

Az assembler további előnye, hogy alkalmazhatunk a BASIC REM utasításhoz hasonló megjegyzéseket a sorok után, pontosvesszővel (;) elválasztva. A pontosvesszőt követő szöveg csak a programozót segíti a program jobb áttekintésében, a fordító figyelmen kívül hagyja.

```
 10 TABL1 =    $1000; első táblázat
 20 TABL2 =    $2000; második táblázat
 30       *=   $3000
 40 KEZD  LDX  #$00; táblázatmásoló eleje
 50 KIIR  LDA  TABL1,X
 60       STA  TABL2,X
 70       INX
 80       BEQ  VEGE
 90       JMP  KIIR
100 VEGE  BRK  ;program vége
```

Ezek után már felírhatjuk egy forrásnyelvű programsor általános felépítését:

sorsz.	cimke	utas.	oper.	megjegyzés
40	KEZD	LDX	#$00 ;	táblázatmásoló eleje

Sorszám: formája és szerepe megegyezik a BASIC sorszámmal.

Cimke v. szimbólum: egyes utasításoknak vagy tárcímeknek adható név, megadása nem kötelező (A szimbólumok hosszát minden assembler egyénileg maximálja, formailag is programonként változik a szabály. Részletesen majd később, egy konkrét assembler, az ASS-16 leírásánál tárgyaljuk.)

Utasítás: a processzor utasításkészletének valamelyike assembly nyelven

Operandus: a gépi utasításhoz kapcsolható operandusok egyike.

Megjegyzés: pontosvesszővel elválasztva tetszőleges szöveg, hosszát csak a programsor lehetséges hossza korlátozza; önálló sorként is szerepelhet.

```
10 ;** táblázatmásoló program **
```

1.6.1. AZ ASS–16 PROGRAM

Az ASS–16 program a C 16-os gépre írt assembler fordítóprogram, de a Plus/4-esen is használható. Minimális eltéréssel megegyezik a C 64-esen futó PROFI-ASS 64 programmal.

Az ASS–16 program betöltése után az assembly forrásnyelvű programot egy normál BASIC programhoz hasonlóan kell begépelni. A sorokat sorszámozni kell és ugyanúgy, mint BASIC-ben, egy sorba több utasítást is írhatunk kettősponttal elválasztva egymástól. Az első nem BASIC utasítás előtt azonban szerepelnie kell egy SYS 12224 utasításnak, mert ez fogja elindítani a fordítót. A BASIC-hez hasonló írásmódnak egy másik nagy előnye, hogy a betöltést és a tárolást a hagyományos BASIC LOAD és SAVE parancsokkal lehet elvégezni.

Ha a programunkat összeállítottuk, a RUN paranccsal indítsuk el. Ekkor a program az esetleges BASIC utasításokat végrehajtja, de az első assembly utasítás előtt szereplő SYS 12224 elindítja a fordítást assemblyből gépi kódra. Az assembly sor általános formátuma az előző részben már szerepelt, azonban az ASS–16-ban van néhány speciális szabály.

sorszám: formában és jelentésben is azonos a BASIC sorszámmal

cimke: megadása nem kötelező, el is maradhat. A sorban címke egymagában sosem állhat, mindig egy utasításnak kell követnie. Alkalmazásakor csak nyolc karakter hosszúságban különbözteti meg a fordító. Mindenképpen betűvel kell kezdődnie, de utána már állhat betű és szám tetszőleges sorrendben.

mnemonik: a 8501-es processzor utasításkészletének egyike assembly nyelven

operandus: a mnemonik által meghatározott címzési módok egyike. Nem csak hexadecimális formában adható meg, sőt egy kifejezés is lehet, melynek az értékét a fordító számítja ki. Ha a cím $FF-nél kisebb, akkor az ASS–16 program automatikusan 0. lapos címzésnek fordítja. Ha az operandus elé felkiáltójelet (!) teszünk, akkor ezeket a $FF-nél kisebb értékű kifejezéseket is kétbyte-os címnek fordítja.

```
SBC $C8,Y = $F5 $C8
SBC !$C8,Y = $F9 $C8 $00
```

Megjegyzés: pontosvesszővel kell kezdeni. Az ezután álló szöveget a fordító nem veszi figyelembe az első kettőspontig. A kettőspont után álló utasítást azonban már lefordítja.

Az ASS–16 program az operandusokat több alakban is kezeli, sőt műveletek eredményeként is keletkezhet az értékük. Az operandusok alakjai a következők lehetnek:

decimális szám:	163	–	LDA	#163
hexadecimális szám:	$C8	–	STA	$C8
bináris szám:	%11101011	–	LDA	#%11101011
ASCII érték:	"C"	–	LDA	#"C"

szimbólum:	TABL1	–	LDA	TABL1
kifejezés:	$3F00+256	–	JMP	$3F00+256
programszámláló (PC):	*	–	BEQ	*+5

Az operandusok értékét műveletek eredményeként is képezhetjük. A kiértékelés balról jobbra történik, amit az esetleges zárójelezés módosíthat. Az ismertetésben a és b számok vagy szimbólumok:

a	+	b	összeadás
a	–	b	kivonás
a	*	b	szorzás
a	!	b	OR
a	&	b	AND
a	↑	b	EOR
a	<	b	a értéke b db bittel balra tolva
a	>	b	a értéke b db bittel jobbra tolva
	<	a	kétbyte-os cím alsó byte-ja
	>	a	kétbyte-os cím felső byte-ja
	!	a	kétbyte-os cím
	–	a	kettes komplemens érték

Direktívák (Pszeudo-kódok)

Direktíváknak nevezzük azokat az utasításokat, amelyeket a forrásnyelvű programban adunk meg a fordítónak. Ezek a direktívák a programban nem jelentenek végrehajtható utasítást, hiszen végrehajtani csak a processzor utasításkészletéből lehet valamelyiket. Szerepük az, hogy a fordítónak adjanak különféle utasításokat a fordítás menetére vonatkozóan. Ezek csak a programozó számára jelentenek segítséget, számolásoktól, táblázatokban való keresgéléstől, egyforma programrészek többszöri begépelésétől mentesítik őt.

A direktívák többnyire ponttal (.) kezdődnek, ez alól azonban van kivétel.

Programszámláló-vezérlő: *= <operandus> vagy <cimke>

Leggyakrabban a program kezdőcímének megadására használják. Ilyenkor egy direkt értéket írnak utána pl. *= $1800. Ha nem adjuk meg, akkor a program $2000-től kerül a tárba. Lehet azonban cimkét is írni, ilyenkor a cimke értékét tekinti a számláló aktuális értékének. Ha pl. előzőleg definiáltunk egy KEZD = $1800 cimkét, majd megadjuk, hogy *= KEZD, akkor a programunk $1800-tól kezdődik. További alkalmazási lehetősége helyfoglalás táblázatoknak.

```
TABL1   *=    $1800   ;első táblázat eleje
TABL2   *=    *+256   ;második táblázat eleje
        *=    *+256   ;program kezdet
        LDA   #0      ;program első utasítása
```

Ezekkel az utasításokkal két 256 byte-os táblázatnak foglaltunk helyet a tárban. Az LDA utasítás már a két táblázat után következik $1A00-tól. Ezt a program-

számláló módosítást tetszőlegesen használhatjuk akár visszafelé is, egyre azonban ügyelni kell; csak akkor használjuk, ha közvetlenül a tárba fordítjuk a programot (l. a későbbi .OPT direktívát). Külső tárolóra való fordításnál ne alkalmazzuk, hiszen a tároló nem tud a programszámláló átállításáról, csak a kezdőcímtől folyamatosan tudja rögzíteni az adatokat.

A programszámláló értéke nemcsak írható, de olvasható is. Rövidebb ugrásoknál, főleg feltételes elágazások használatakor cimkéket takaríthatunk meg a * szimbólumként való használatával:

```
10   TABL1  *=   $1800
20   TABL2  *=   *+256
30          *=   *+256
40   KEZD   LDX  #$00
50   KIIR   LDA  TABL1,X
60          STA  TABL2,X
70          INX
80          BEQ  VEGE
90          JMP  KIIR
100  VEGE   BRK
```

Ebben a programrészben összesen 5 cimke szerepel. Tegyük fel, hogy ez egy olyan hosszú program része, hogy a sok cimke nem fér el a szimbólumtáblázatban (l. részletesen a .SYM direktívánál) és egy cimkét meg kellene takarítani. Ezt lehet a * felhasználásával elérni.

```
10   TABL1  *=   $1800
20   TABL2  *=   *+256
30          *=   *+256
40   KEZD   LDX  #$00
50   KIIR   LDA  TABL1,X
60          STA  TABL2,X
70          INX
80          BEQ  *+5
90          JMP  KIIR
100         BRK
```

Ebben a példában a VEGE cimkét takaríthattuk meg, mert itt egy rövid ugrás szerepelt. Mivel a BEQ feltételes ugrásnak egyetlen utasítást kellet átugornia, így nem kellet sokat számolnunk. A 80-as sorban a * programszámláló a BEQ helyére mutat. Nekünk át kell ugrani a BEQ operandusát (*+1), a JMP utasítást (*+2), a JMP kétbyte-os operandusát (*+4), és ráállni a programszámlálóval a BRK utasításra, ami a JMP kétbyte-os operandusa után van (*+5).

Értékadó-jel: ← (balra nyíl)

Az assembler programoknál már említettük, hogy egy cimkének pl. a TABL1 = $1800 utasítással lehet értéket adni. Ekkor azonban az újradefiniálás már nem lehetséges. Ellenben, ha ezzel a jellel adjuk meg, akkor újradefiniálhatjuk a szimbólum értékét, természetesen ismét ezzel a jellel.

```
...
 20  TABL1   = $1800
 30  TABL2   *= *+256
...
 70  SZAML   ← $C7                   ;számláló alapértéke
 80    LDA   TABL1      +SZAML       ;$18C7 tartalma AC-ban
 90    STA   TABL2      +SZAML       ;AC tartalma $19C7-be
100  SZAML   ←          SZAML+1      ;SZAML újradefiniálása
110    LDA   TABL1      +SZAML       ;$18C8 tartalma AC-ban
120    STA   TABL2      +SZAML       ;AC tartalma $19C8-ba
...
```

A fordítás során az LDA és STA utasítások után lévő operandus értékét a módosított SZAML érték határozza meg.

Pont direktívák

A következőkben a ponttal (.) kezdődő direktívákat ismertetjük. A négybetűs direktívák megkülönböztetése is csak az első három betű alapján történik, tehát elég az első három betűt beírni.

.BYT(E): a direktíva után álló egy vagy több kifejezés értéke a tár aktuális címére kerül.

```
*= $1800

.BYTE $FF,254,%01111111,"a"
```

Példánk a $1800 címtől kezdve elhelyezi a tárban a .BYTE után felsorolt négy értéket, tehát a $FF, a $FE, a $7F, és a $41 számokat.

.ASC: a direktíva után idézőjelben szereplő szöveg ASCII kódja kerül a tár adott részére.

```
*= $1800

.ASC "ASS-16"
```

Az "ASS-16" szöveg ASCII kódjait helyezi el a tárban $1800-tól.

.WOR(D): a direktíva után álló kifejezések kétbyte-os címként kerülnek a tárba, tehát fordított sorrendben (alsó byte, felső byte).

Például a .WORD $FFD2 kifejezés a tárba $D2, $FF-ként kerül beírásra.

.GOT(O): feltétel nélküli ugró utasítás. Az ugrási cím a direktíva után álló sorszám. Ez nem azonos a JMP utasítással, mert itt a fordítónak adjuk meg, hogy hányas sorszámú utasításnál folytassa a fordítást.

.IF: feltételes elágazást biztosít, ismét csak a fordító számára. Ha a .IF után álló kifejezés értéke nem 0, akkor az utána kettősponttal elválasztva szereplő utasítást fordítja az assembler, ellenkező esetben a következő sor utasítását.

Ez utóbbi két direktíva (.GOTO és .IF) ciklusszervezésre használható. Ez a ciklus azonban nem azonos egy programciklussal, csak fordítási ciklust eredményez.

Fordítás előtt	Fordítás után
10 TABL1 *= $1800	1500 LDA $1803
20 TABL2 *= *+256	1503 STA $1903
30 *= *+256	1506 LDA $1802
40 SZAML ← $ 03	1509 STA $1902
50 * = $1500	150C LDA $1801
60 LDA TABL1+SZAML	150F STA $190
70 STA TABL2+SZAML	1512 BRK
80 SZAML ← SZAML-1	
90 .IF SZAML: .GOTO60	
100 BRK	

A példaprogramunk fordítása, amint látható, nem progamciklust, hanem fordítási ciklust eredményez. A fordítás során háromszor kerül a tárba az LDA STA ciklusmag, pedig csak egyszer írtuk le. A fordító tehát a .IF utasítástól függően ugrik vissza a 60. sorba, és ott folytatja a fordítást. Ezzel a módszerrel begépelést takaríthatunk meg abban az esetben, ha több hasonló programrészt kellene egymás után beírni.

A .IF és a .GOTO direktívákkal tehát egy fordítási ciklust szervezhetünk, ami tulajdonképpen többször fordítja le a ciklusmagban szereplő programrészt.

.SYM: az ASS-16 program a szimbólumtáblázatot $2FCO-tól lefelé tárolj a tárban. Ez a táblázat maximum a forrásnyelvű program tetejéig mehet le, hiszen ha túlmegy, akkor belelógna magába a forrásprogramba. A .SYM direktiva után álló operandus a szimbólumtáblázat alsó határcímét állítja be.

.SYM $2000

.LIN(K): a fordítás során forrásnyelvű programok összefűzésére szolgál.

Formátuma: .LINK <egységszám>,"file-név"

Az egységszám kazettánál 1, lemezegységnél 8. Ha a fordító fordítás közben ezzel az utasítással találkozik, az <egységszám>-ban megadott egységről betölti a "file-név" alatt meghatározott forrásnyelvű programot és ezzel folytatja a fordítást. Többszörös összefűzés is megengedett. Lemezegységnél ez mindenkor alkalmazható, tehát tárba fordításkor és lemezre való forditáskor egyaránt. Kazettás egységnél csak a tárba fordításkor célszerű alkalmazni (l. .OPT direktíva).

.OPT: a fordítóprogram fordítását, listázását szabályozza.

Formátuma: .OPT <opció>,<opció>

P opció: a fordítóprogram listázását szabályozza. Ha csak a képernyőre szeretnénk a fordítási listát, akkor ezt a .OPT P utasítással érhetjük el. Ha nyomtatóra szeretnénk listázni, akkor előzőleg BASIC-ből meg kell nyitni a nyomtatót, majd a P opció után kell írni a logikai file-számot.

```
10  OPEN    2,4    ;nyomtató megnyitása
20  SYS   12224    ;fordító indítása
30   OPT     P2    ;lista nyomtatóra
```

O opció: meghatározza, hogy a fordítás során a kész gépi kódú program hová kerüljön.

A következő formákban alkalmazható:

OO: a pogram a *= direktívában megadott helyre kerül. Ügyelni kell arra, nehogy a forrásprogram, vagy a fordító helyére írjunk.

```
10  OPEN     2,4    ; nyomtató megnyitása
20  .OPT   P2,OO    ;lista nyomtatóra, program *= szerint
                    ;a tárba kerül
30  *=     $1500    ;kezdőcím $1500
```

O: a program a forrásprogram utáni szabad területre kerül, melynek a kezdőcímét a $2D és $2E címen lévő kétbyte-os mutató határozza meg. A *= direktíva azonban nem marad figyelmen kívül. A fordítás ugyanis aszerint történik, mintha a * = -ben megadott helyre kerülne a program. Ily módon elérhető, hogy olyan programot fordítsunk, ami majd pl. az ASS–16 fordító helyén fog működni. Fordítás után monitor T parancsával a helyére kell másolni a kész programot.

O<szám>: a program egy OPEN-nel megnyitott file-ba kerül. A logikai file-számot az O után kell írni. A kívánt file megnyitását az előző példában leírt módon BASIC-ben kell elvégezni.

Az előzőekben tárgyalt O ill. OO opciók mind a tárba fordítás módját határozták meg. Ezek azonban csak rövidebb programoknál használatosak a szabad tárkapacitás szűkössége miatt. Az ASS–16 program $2FC0-tól felfelé, $4000-ig helyezkedik el, a szabad tár tehát csak $1000-tól 2FC0-ig tart, azonban ez sem teljesen szabad. A program a szimbólumtáblázatot $2FC0-tól lefelé tárolja, tehát ez is csökkenti a szabad tárkapacitást, így egy kellően bonyolult forrásnyelvű program el sem fér a tárban egyszerre, nemhogy még a tárgykód is a tárba kerüljön. Erre találták ki a külső tárolóra való fordítást. Ekkor a fordítás nem a tárba, hanem egy külső, előzőleg megnyitott file-ba történik.

Formátuma: O<log.file-szám>

```
10  OPEN  2,8,2,"PRÓBA,P,W"
20  SYS   12224
30  .OPT  P,O2
```

Példánkban először megnyitottuk a lemezegységet 2-es logikai file-számon PRÓBA file-név alatt. A 20-as sorban indítottuk el a fordítást, majd a 30-as sorban utasítottuk a fordítót, hogy a listát a képernyőre, a tárgykódot a PRÓBA nevű fileba írja. Kazettás egységnél ugyanígy kell elvégezni mindent, csak az egységszám 1. Ekkor azonban a tárgykódot csak egy külön programmal lehet visszatölteni a kazettáról a tárba. Ezt a programot az ASS–16-hoz kazettán mellékelik.

N opció: az előzőekben megadott .OPT opciók törlődnek. Ha valamelyiket ismét aktivizálni szeretnénk, újabb .OPT utasítást kell kiadnunk.

```
9000  .OPT N      ;eddigi opciók törlése
9010  .OPT OO,P   ;új opciók
```

.BAS: fordítás közben ezzel a direktívával adhatjuk vissza a vezérlést a BASIC interpreternek. Az ezután következő BASIC utasításokat az interpreter értelmezi és végrehajtja, majd az újabb assembly utasítás előtt álló SYS 12230 utasítás ismét elindítja a fordítót. Ez elég ritkán használt megoldás.

.END: ez jelzi a fordítóprogramnak a forrásnyelvű program végét. Ekkor a fordítás befejeződik, a vezérlés visszakerül BASIC-be. A tárgykód vagy a tárban vagy egy külső file-ban (.OPT szerint) készen áll.

A .END <egys.szám>,"kezdő file-név" formátum esetén a .LINK utasítás végét jelezzük, tehát azt, hogy befejeződött az összefűzés. A "kezdő file-név" nevű forrásnyelvű program visszatöltődik.

A használható SYS utasítások összefoglalása

SYS 12224 : a fordítás elindítása. Az első sor begépelése előtt célszerű kiadni, mert az interpreter tokenizálási (2.8. alfejezet) rendszerét is módosítja. Ezen kívül kizárja a BASIC területből a $2FC0-tól fölfelé eső részt, így megvédi a fordítót attól, hogy a forrásnyelvi programmal felülírjuk.

SYS 12227 : az interpreter tokenizálási rendszerét visszaállítja alapállapotba. A módosított tokenizálási rendszerben bevitt sorok ezután nem listázhatók.

SYS 12230 : a .BAS után ezzel folytathatjuk a forrásnyelvű program fordítását.

SYS 12233 : a NEW parancs vagy a RESET gomb megnyomása után visszanyerhetjük a forrásnyelvű programunkat.

Hibaüzenetek

A fordítás során, ha valamit nem jól írtunk, a fordító hibaüzenetet ad. A SYNTAX hibaüzenet előtt álló szám utal a hiba okára:

0 – címke, utasítás nélkül
1 – érvénytelen műveleti kód
2 – érvénytelen címzési mód
3 – nem megengedett karakter
4 – páratlan számú zárójel
5 – érvénytelen kifejezés
6 – hiányzik egy vessző
7 – érvénytelen direktíva
8 – érvénytelen szimbólum
9 – a műveleti kód címzése nem megengedett

ILLEGAL QUANTITY – a kifejezés értéke a megengedett határokon kívül esik

OVERFLOW – túlcsordulás, túl hosszú sor

BRANCH OUT OF RANGE – a relatív ugrás nagysága nagyobb, mint 127 byte

REDEFINITION – újradefiniálás az egyenlőségjellel

UNDEF'D STATEMENT – cimke nincs definiálva, vagy a .GOTO nemlétező sorra mutat

SYM TABLE OVERFLOW – a szimbólumok nem férnek el a rendelkezésre álló helyen

OUT OF MEMORY – a tárgykód rendelkezésére álló hely kicsi (.OPT O módban)

DEVICE NOT PRESENT – úgy hivatkoztunk egy külső egységre, hogy az nem elérhető

IEEE – egyéb hiba az IEC buszon

DISK – lemezhiba

2. FEJEZET
GÉPI KÓDÚ PROGRAMOZÁS HALADÓKNAK

2.1. Gépi programok és rutinok használata

A teljes egészében gépi nyelven írt program viszonylag ritka. Gyakran fordul elő az, hogy BASIC programokat támogatnak gépi rutinokkal. Ilyenkor problémát okozhat a rutin megfelelő elhelyezése, de gondot jelenthet az is, ha a gépi és a BASIC program között adatátadásra is szükség van.

A gépi kódú programozás témakörébe tartoznak az operációs rendszerrel kapcsolatos manipulációk is. Ilyen mindennapos manipuláció például a saját karakterkészlet használata. A BASIC interpreter és a KERNAL módosítása, kiegészítése gyakran alkalmazott programozói fogás. Ezek részletes ismertetésére később kerül sor.

Gépi program hívása BASIC-ből

A legkézenfekvőbb indítási mód a SYS utasítással való hívás. Ez esetben a gépi és BASIC program kapcsolatának legkényesebb pontja a paraméterátadás. Ez legegyszerűbben a közösen használt tárterületeken keresztül oldható meg, melyeket a BASIC a POKE és PEEK utasításokkal kezelhet. Bonyolultabb megoldás az, ha a gépi program egy BASIC változó értékét írja vagy olvassa. Ez utóbbihoz ad segítséget a BASIC programok és változók tárolásáról szóló fejezet.

A SYS használatához feltétlenül tudni kell, hogy a gépi rutin indítása előtt a regiszterek átveszik a \$07F2–\$07F5 területre beírt értékeket.

AC: 07F2
XR: 07F3
YR: 07F4
SR: 07F5

A BASIC-be való visszatérés előtt a regiszterértékek erre a területre íródnak ki. Ez a terület tehát kiválóan alkalmas mindkét irányú paraméterátadásra.

Egy ritkább megoldás az, amikor az átadandó érték a SYS ugrási címe után szerepel vesszővel elválasztva. A meghívott gépi program a BASIC programterületről olvassa a paramétert a CHRGET rutin segítségével. (A rutin leírása a ROM-rutinok jegyzékében található. Helyette használható más beolvasó rutin is.)

Gépi rutin hívására alkalmas BASIC utasítás az USR függvény is. Működési mechanizmusa a következő:

1) Az USR utáni zárójelben levő kifejezés értéke a lebegőpontos akkumulátorba (leírása a 2.3. alfejezetben) töltődik.

2) Vezérlésátadás történik a \$0501–\$0502 mutató által megadott címre (gépi rutinra). Értékét a felhasználó állítja be.

3) A felhasználói rutin lefut, felhasználva (vagy figyelmen kívül hagyva) a lebegőpontos akkumulátor értékét.

4) Visszatéréskor a BASIC az USR függvény értékének a lebegőpontos akkumulátor aktuális értékét veszi.

Mindezekből látható, hogy az USR elsősorban a matematikai jellegű rutinok hívását segíti.

Mind a SYS, mind pedig az USR használatánál igaz, hogy a gépi rutinokat RTS-sel kell lezárni. Hatására a BASIC program a hívó utasítás után folytatódik. Addig azonban, amíg a fejlesztést vagy tesztelést monitorból végezzük, legcélszerűbb a rutint BRK-val lezárni, és csak a fejlesztés legvégén kicserélni RTS-re.

A gépi program elvileg bármely RAM területen futhat, de ügyelni kell arra, hogy az esetleges BASIC programot ne zavarja meg. Erre egy elég biztos módszer az, ha a rendszerváltozók területén lévő nagyobb üres helyekre írjuk programunkat (leírásuk a 2.10. alfejezetben). Nagyobb szabad területet teremthetünk a BASIC RAM végén, ha a "BASIC terület vége" (\$37–\$38) mutatót alacsonyabbra állítjuk. Ide szokás például a módosított karakterkészletet is tenni. A "BASIC program vége" mutató (\$2D–\$2E) magasabbra állításával az utolsó BASIC sor végét jelző 3 darab 0, és a változóterület kezdete közötti rész is szabaddá válik. Ez a terület azonban egy új BASIC sor beírásakor feljebb tolódik.

A Plus/4-esnél és a bővítős C 16-osnál tudni kell azt, hogy ha \$8000 feletti címet hívunk SYS-szel, akkor a ROM-ban lévő program, tehát a rendszerprogram fog futni! A POKE és PEEK viszont mindig a RAM-ra vonatkozik. Ezzel kapcsolatban célszerű a memórialapozásról szóló fejezetet áttanulmányozni.

Rövidebb gépi rutint szokás BASIC programból a tárba írni. Ilyenkor a gépi program kódját DATA-sorokból olvassuk ki, és POKE utasítással írjuk a megfelelő helyre.

2.2 Grafikus üzemmódok programozása

2.2.1. NORMÁL, KARAKTERES ÜZEMMÓD

A C 16-os bekapcsolás után ebben az üzemmódban jelentkezik be. Ilyenkor a képernyő 25 sorra, soronként pedig 40 oszlopra van felosztva, tehát a képernyő összesen 1000 karaktert tartalmazhat. A képernyőmemóriába írt értéktől függ, hogy a hozzá tartozó képernyőhelyen milyen karakter jelenik meg. A megjelenő karakter színét a színmemória, alakját pedig a karakter ROM határozza meg.

Képernyőmemória

Ebben a tárrészben a képernyő minden egyes karakterhelyének egy byte van kijelölve, ez az összesen 1000 byte a képernyőmemória. Elhelyezkedése a $0C00–$0FFF-ig terjed, ebből azonban csak $0FE7-ig tart a látható terület (l. 1. táblázat). Ha a képernyőn akarunk megjeleníteni egy karaktert, akkor a karakter kódját erre a tárterületre kell beírni. Ezek a képernyőkódok csak 0-tól 127-ig ($7F) határoznak meg különböző karaktereket, mert a legfelső, a 7. bit az inverz megjelenítés kapcsolója. Ha értéke 1, a karakter inverz módon fog megjelenni.

A képernyőkód-táblázat tartalmazza a képernyőn megjelenő karakterek kódját (l. 3. táblázat).

FIGYELEM: ez nem azonos az ASCII kódtáblázattal !

Próbáljuk ki a következő programot:

```
LDA #$01  ; az "A" betű kódja
STA $0C00 ; a képernyő bal felső sarka
BRK
```

A program lefuttatása után a képernyő bal felső sarkában megjelenik egy A betű.

Színmemória

A színmemóriában lévő információ a képernyőn megjelenő karakterek színét, fényerejét és villogását határozza meg. A tárban a $0800–$0BFF területet foglalja el, azonban itt is csak $0BE7 az utolsó látható hely (l. 2. táblázat). A 0.–3. bit a karakterhely színét, a 4.–6. bit a fényerejét állítja be. A 7. bit 1-re állításával a karakterhely villogását kapcsolhatjuk be.

Ha az előző példában szereplő A betűt egy adott színnel akarjuk megjeleníteni, akkor a hozzá tartozó színmemóriát is állítanunk kell:

```
LDA #$01
STA $0C00
LDA #$71 ; 7-es fényerejű fehér
STA $0800; a bal felső sarok színe
BRK
```

Ha lefuttatjuk a programunkat, akkor a bal felső sarokban egy fehér A betű fog megjelenni. Ügyeljünk arra, hogy a háttér ne fehér legyen, mert akkor nem látunk semmit!

Ha ezt az A betűt villogtatni akarjuk, akkor a 7. bitet is magasra kell állítani.

```
LDA #$F1  ; 7-es fényerejű villogó fehér
STA $0800
BRK
```

Karakter ROM

Minden karakter alakja egy 8×8-as pontmátrixban van leírva. A pontok ki- vagy bekapcsoltsága egy bit alacsonyra vagy magasra állításával történik. A karakter egy sora egy byte (8 bit) információt tartalmaz, tehát egy teljes karakter leírása 8 byte-ot vesz igénybe. Az a tárterület, ahol a karakterek alakját rögzítették, a $D000–$D7FF-ig tart. Ez ROM, közvetlen felülírása nem lehetséges!

A 0 képernyőkódú @-jel bittérképe a $D000–$D007 területen van leírva. Értelmezését a következő ábra szemlélteti:

Cím	Karakter	Bináris	Hexa
D000	0000	00111100	3C
D001	00 00	01100110	66
D002	00 000	01101110	6E
D003	00 000	01101110	6E
D004	00	01100000	60
D005	00 0	01100010	62
D006	0000	00111100	3C
D007		00000000	00

$D008-tól $D00F-ig pedig az A betű leírása következik, ugyanilyen felépítésben.

$D000-tól $D3FF-ig tart a nagybetű/grafikus karakterkészlet karaktereinek képi információja.

$D400-tól $D7FF-ig található a kisbetű/nagybetű üzemmód karakterkészlete.

Tehát, ha a nagybetű/grafikus üzemmódban vagyunk, akkor a TED chip a $D000-t veszi kiindulásnak, és onnan számítja a 128 karakterhez tartozó információt. Kisbetű/nagybetű módban ez a kiindulás felkerül $D400-ra.

A TED chip $FF07 című regiszterének a 7. bitje az ún. RVS kapcsoló. Ha ez a bit alacsony, akkor a színmemória 7. bitjének bekapcsolása inverz karakter megjelenését eredményezi (ez a normális üzemmód). Ha a $FF07 7. bitjét magasra (1-re) állítjuk, akkor az inverz megjelenítés a képernyőmemória 7. bitjével nem lehetséges, a 7. bit is a karakter kódjához fog tartozni. Ily módon lehetőség nyílik 256 különböző karakter egyidejű használatára.

Saját karakterkészlet

Magától értetődik, hogy a karakter ROM nem írható felül, tehát saját tervezésű karakterek alapállapotban nem alkalmazhatók. Néhány bit módosításával azonban elérhető, hogy az általunk definiált karakterkészlet jelenjen meg a képernyőn.

Ha a TED chip $FF12-es regiszterének 2. (4-es helyi értékű) bitje 1, akkor a karakterkészlet a ROM-ban, ha 0, akkor RAM-ban van. A $FF13-as regiszter 2.–7. bitjei pedig a karakterkészlet helyét adják meg K-ban. Tehát ha a saját karakterkészletünk pl. $3800-tól kezdődik, akkor a $FF12 címen a 2. bitet alacsonyra kell állítani. Itt eredetileg $C4 van, amit $C0-ra kell módosítani. Ezután a $FF13 regiszterben kell megadni a kezdőcímet, $38-at (itt eredetileg D1 van). Ezeket a módosításokat a monitorból a legegyszerűbb végrehajtani.

A következőkben leírt példa egy karakter módosítását tartalmazza. Ez azokban az esetekben mindig alkalmazható, ha a meglévő karakterkészletet nagyrészt változatlanul akarjuk használni, és csak néhány karaktert szeretnénk módosítani.

Első lépésben monitorból másoljuk ki, azaz töltsük át a karakterkészletünket a ROM-ból a RAM-ba, pl. $3800-tól:

```
T D000 D7FF 3800
```

Ekkor a teljes karakterkészlet, tehát a nagybetű/grafikus és a kisbetű/nagybetűs készlet is áttöltésre kerül. Helytakarékossági megfontolásból, ha biztosak vagyunk benne, hogy csak az egyik készletet használjuk, akkor elég csak azt átmásolni, így a RAM helyszükséglet a felére csökken.

Következő lépésünk a $FF12, $FF13 címek átállítása lesz a már említett módon monitorból:

```
>FF12 C0 38
```

Programból:

```
LDA #$C0   ; új érték
STA $FF12  ; FF12-be, karakterkészlet RAM-ban
LDA #$38   ; új érték
STA $FF13  ; FF13-ba, karakterkészlet helye
BRK
```

Ezek után $3800-tól rendelkezésünkre áll a módosítható karakterkészlet. Próbáljuk is ki, módosítsuk pl. @ -jelet egy emberalakra. A @ képernyőkódja 0, tehát ez az első karakter. Alakja a $3800–$3807 tárterületen van kódolva, nekünk ezt kell módosítani monitorból:

```
>3800 18 18 FF 18 18 24 42 81
```

Ha mindezt végrehajtottuk, akkor @-billentyű lenyomásakor a mi emberalakunk fog megjelenni. Ha később is használni szeretnénk a karakterkészletünket, akkor érdemes eltárolni, hogy ne kelljen minden alkalommal ezt a hosszadalmas utat végigjárni. Monitorban gépeljük be a következő parancsot:

S "KARAKTER" 1 3800 4000

Ha már szalagon megvan a karakterkészletünk, akkor később egyszerűen monitorból betölthetjük, és a mutatók átállításával ($FF12–$FF13) újra használhatjuk.

2.2.2. BŐVÍTETT HÁTTÉRSZÍN ÜZEMMÓD

Ebben az üzemmódban minden egyes karakterhelyre külön lehet állítani a háttér színét. Ha pl. a teljes hátterünk piros, akkor megjeleníthetünk rajta egy sárga A betűt kék háttérrel és mellette egy zöld B betűt fehér háttérrel.

Mielőtt azonban átállítanánk a bővített háttérszín üzemmód kapcsolóját, be kell állítanunk az általa használt színeket és kötnünk kell egy kis kompromisszumot. Ebben az üzemmódban ugyanis egyszerre csak 64 karakter használható (0–63). A képernyőkód 6.–7. bitjei ugyanis azt a regisztert határozzák meg, amelyikből a TED-nek vennie kell a háttérszín-információt. A két bittel összesen 4 regisztert lehet kiválasztani $FF15-től $FF18-ig. Ebben a négy regiszterben kell megadni a négyféle háttér színét és fényerejét. A megjelenő karakter színét és fényerejét továbbra is a színmemória határozza meg. Ezek után már konkrét példán keresztül vizsgáljuk meg részletesebben ezt az üzemmódot.

A karakterkód 6.–7. bitjei és a hozzájuk tartozó színregiszterek közötti összefüggést a következő ábra szemlélteti:

6.–7.bit	Regiszter
00	$FF15
01	$FF16
10	$FF17
11	$FF18

Az előbbiekben már utaltunk rá, hogy először össze kell állítani egy elfogadható színkombinációt a színforrás-regiszterekben és a színmemóriában. Először tehát válasszuk ki a négy színt, amit háttérként akarunk alkalmazni, és írjuk be a megfelelő regiszterekbe.

```
>FF15 DE E2 E5 F7
```

Programból ugyanez

```
LDA  #$DE   ;teljes háttér színe
STA  $FF15  ; háttérszín regisztere
LDA  #$E2   ; 1. háttérszín
STA  $FF16  ; a regiszterben
```

```
LDA  #$E5   ; 2. háttérszín
STA  $FF17  ; a regiszterben
LDA  #$F7   ; 3. háttérszín
STA  $FF18  ; a regiszterben
```

Ha valamelyik módszerrel elvégeztük a beállítást, akkor a teljes háttérszín kék lesz (ezt már látjuk is), az 1-es háttérszín piros, a 2-es háttérszín zöld, a 3-as pedig sárga (ezek még nem jelennek meg, hiszen nem kapcsoltunk át bővített háttérszín üzemmódra). Ezután állítsuk be a képernyő bal felső sarkához tartozó négy egymás melletti színmemóriát ($0800–$0803) mondjuk feketére:

```
>0800 00 00 00 00
```

Programmal:

```
2000  LDX  #$00     ; index kezdőérték
2002  TXA           ; akkumulátorba is 0
2003  STA  $0800,X  ; a négy karakterhely
2006  INX           ; indexérték növelése
2007  CPX  #$04     ; elérte a 4-et?
2009  BNE  $2003    ; ha nem vissza
200B  BRK           ; ha igen, vége
```

Ezzel beállítottuk, hogy a megjelenő karakterünk mind a négy helyen fekete legyen. Ezután már csak az van hátra, hogy megjelenítsük a karaktereket a bal felső sarokban, majd átkapcsolva bővített háttérszín üzemmódra megnézzük, milyen színeket sikerült összeállítanunk.

A karakterek megjelenítésekor ügyeljünk arra, amit a karaktermód 6.–7. bitjei és a színforrás-regiszterek közötti összefüggésről az előbbi táblázatunk tartalmaz. A bal felső sarokban megjelenő I betű legyen normál háttérszínű. Ekkor a 6.–7. bitnek értéke 00. Mellette jelenjen meg a fekete I betű az 1-es háttérszínnel, tehát a karakterkód 6.–7. bitjei 01. A harmadik megjelenő I betű a 2-es háttérszínt használja, felső bitjei 10, míg a negyedik karakter 11 állású felső bitjei a 3-as definiált háttérszín megjelenését eredményezik.

Az I betű képernyőkódjai bővített háttérszín üzemmód esetén a négyféle háttérszínnel:

Színforrásregiszter		Képernyőkód
normál	$FF15	00 001001=$09
1-es	$FF16	01 001001=$49
2-es	$FF17	10 001001=$89
3-as	$FF18	11 001001=$C9

Ezt a négy háttérszínű I betűt írjuk be a képernyőmemóriába $0C00-tól:

```
>0C00 09 49 89 C9
```

Programból:

```
LDA  #$09    ; normál háttérszínű „I"
STA  $0C00   ; a bal felső sarokban
LDA  #$49    ; 1. háttérszínű „I"
STA  $0C01   ; a második pozícióban
LDA  #$89    ; 2. háttérszínű „I"
STA  $0C02   ; a harmadik pozícióban
LDA  #$C9    ; 3. háttérszínű „I"
STA  $0C03   ; a negyedik pozicióban
```

Ha ezt elvégeztük, akkor még nem jelenik meg különböző színnel a négy karakterünk, mert még nem kapcsoltunk át a bővített háttérszín üzemmódra. Ezt a TED $FF06 címen levő regiszter 6. bitjének magasra állításával érhetjük el. Eredeti értéke $1B, ezt $5B-re kell módosítani.

```
>FF06 5B vagy

LDA  #$5B
STA  $FF06
```

Ekkor feltétlenül $5B-t írtunk a tárcímre, függetlenül attól, hogy előtte mi volt ott. Ha az összes bitet változatlanul akarjuk hagyni, csak a mi 6. bitünket szeretnénk magasra állítani, akkor célszerűbb (és elegánsabb) az ORA utasítást használni:

```
LDA  $FF06   ; FF06 tartalmát az akkumulátorba
ORA  #$40    ; 6. bit magasra állítása
STA  $FF06   ; visszatenni az FF06 címre
```

Ezek után már látni kell a bal felső sarokban a négy I betűt. Az első a normál kék háttérrel, a második piros, a harmadik zöld, a negyedik pedig sárga háttérben jelenik meg. Ha ezután még az I betűk színét is különbözőre akarjuk változtatni, akkor a színmemóriát kell módosítani a következőképpen:

```
>0800 F1 00 C4 C6
```

Ez egy fehér, egy fekete, egy lila és egy kék I betűt eredményez a négyféle háttérszínen.

2.2.3. TÖBBSZÍNŰ KARAKTER ÜZEMMÓD

A karakteres üzemmódoknak ez a leglátványosabb, de talán a legbonyolultabb része. Ebben az üzemmódban ugyanis lehetőség nyílik arra, hogy a kepernyőn megjelenő karakterünk négy különböző színből tevődjön össze. Ez a négy szín a háttérszín ($FF15), a színmemóriában tárolt szín, valamint a bővített háttérszín üzemmódból már ismert két segédszín-regiszterben a $FF16–$FF17 címeken tárolt szín. Ebben az üzemmódban használhatjuk az összes karaktert, de kompromisszumra itt is szükség van. A vízszintes felbontás ugyanis a felére csökken, tehát egy karakter csak 4 x 8 pontból áll. Ennek az az oka, hogy a karakter ROM-ban a karakter pontjainak színét vezérlő bitek közül az egymás mellettiek párosával "összekapcsolódnak". Így a két bit (azaz egy bitpár) négyféle állapotával lehet meghatározni a megjelenő pont színét. A bitpárok állapota és a színforrás-regiszterek közötti összefüggést a következő leírás szemlélteti:

00 – $FF15 háttérszín regiszter
01 – $FF16 1. segédszínregiszter
10 – $FF17 2. segédszínregiszter
11 – a karakterhelyhez tartozó színmemória

A 0-ás képernyőkódú @ karakter bittérképe most is a $D000–$D007 területen van megadva, azonban jelentése ebben az üzemmódban más.

Cím	Karakter	Bináris	Hexa
D000	0 3 3 0	00 11 11 00	3C
D001	1 2 1 2	01 10 01 10	66
D002	1 2 3 2	01 10 11 10	6E
D003	1 2 3 2	01 10 11 10	6E
D004	1 2 0 0	01 10 00 00	60
D005	1 2 0 2	01 10 00 10	62
D006	0 3 3 0	00 11 11 00	3C
D007	0 0 0 0	00 00 00 00	00

Az ábrából jól látszik, hogyan határozza meg a két egymás mellet álló bit a megjelenő pont színét. Ha egy pontot ki akarunk kapcsolni (azaz háttérszínnel akarunk megjeleníteni), akkor a hozzá tartozó bitpárost 00-ra kell állítani. Ekkor a $FF15-ben definiált háttérszínű lesz a pont. Ha a bitpárt 01-re állítjuk, akkor a $FF16-ban megadott színnel fog megjelenni, míg az 10 a $FF17-ben lévő színt eredményezi. Az ezen a három helyen (tehát $FF15, $FF16, $FF17) lévő színek rögzítettek, beállításuk csak egyszer lehetséges. Ezután minden olyan raszter pontjai, amelyeket 00, 01, 10 bitpárokkal adtunk meg, ezeket a színeket fogják használni. A 11 bitpáros ettől eltér, ez ugyanis karakterhelyenként változhat. Ilyenkor a TED a színmemóriából ($0800–$0BFF) veszi a ponthoz tartozó színinformációt. A 11 bitpárhoz tartozó pont színét tehát a karakterhelyhez tartozó színmemóriában kell

megadni. A színmemóriának még egy lényeges funkciója van. Ezzel lehet karakterhelyenként külön-külön be- vagy kikapcsolni a többszínű karakter üzemmódot. Ez azt jelenti, hogy amelyik karakterhelyhez tartozó színmemória 3. (8-as helyiértékű) bitje magas, az a karakterhely többszínű karakter üzemmódban van. Amelyiknél alacsony, az normál karakteres üzemmódban jelenik meg. Ennek van egy előnye és egy hátránya is. Előnye, hogy a képernyőn lehet keverni a normál és a többszínű karakter üzemmódot, hátránya, hogy a színmemóriában a harmadik bit foglaltsága miatt (ami a színek definiálásában játszik szerepet), csak az első nyolc szín alkalmazható. A fényerőbitek változatlanok.

A többszínű karakter üzemmódra való áttérést a TED $FF07 címen lévő regiszter 4. bitjének magasra állításával érhetjük el. Ekkor azonban még nem látunk semmit, kivéve azokon a helyeken, ahol a színmemória harmadik bitje 1. Tehát ha az egész képernyőt ebben az üzemmódban akarjuk működtetni, akkor a teljes színmemóriát ennek megfelelően kell beállítani.

A színmemória bitjeinek hatása ebben az üzemmódban a következő:

7. bit	: hatástalan, ebben az üzemmódban a kurzor sem villog
6.–4. bit	: fényerő
3. bit	: többszínű üzemmód (ha értéke: 1)
2.–0. bit	: színkód

Ha többszínű karakter üzemmódban kívánunk dolgozni, célszerű ezt saját karakterkészlettel végezni, mert az eredeti ebben az üzemmódban olvashatatlan. Ügyeljünk a $FF16 és a $FF17 regiszterek helyes színbeállítására is.

2.2.4. NORMÁL BITTÉRKÉPES ÜZEMMÓD

Ebben az üzemmódban lehetőség nyílik arra, hogy a képernyő 320*200 pontját külön-külön be vagy ki tudjuk kapcsolni. Ilyenkor ugyanis minden képponthoz a bittérkép által lefoglalt tárterület 1-1 bitje (320*200 bit = 8000 byte) van hozzárendelve. A bitek elhelyezkedése azonban nem sorfolytonos, hanem a karakteres kijelzés analógiáját követi. A bittérkép címei, és a hozzájuk rendelt képpontok elrendezését a következő ábra szemlélteti.

2000 2001 2002 2003 2004 2005 2006 2007	2008 2009 200A 200B 200C 200D 200E 200F	2130 2131 2132 2133 2134 2135 2136 2137	2138 2139 213A 213B 213C 213D 213E 213F
2140 2141 2142 2143 2144 2145 2146 2147	2148 2149 214A 214B 214C 214D 214E 214F	2270 2271 2272 2273 2274 2275 2276 2277	2278 2279 227A 227B 227C 227D 227E 227F

A bittérkép első byte-jának ($2000) 8 bitjével vezérelhető az első pontsorban az első 8 képpont. A második byte alatta, a második sorban elhelyezkedő első 8 képpontot kapcsolja ki vagy be. Ez így megy egészen a 8. byte-ig, ami a 8. sorban kezdődik. A kilencedik byte már szintén az első pontsorban, de a második 8 képpontot kapcsolja. A 9. pontsor első 8 képponja a 320. ($2140) byte-tal kapcsolható.

BASIC-ben a grafikus kép bittérképe $2000–$3FFF tárterületet foglalja el. A pontok fényerejét meghatározó fényerőmemória $1800-$1BFF-ig, míg a színmemória $1C00-$1FFF-ig tart.

Ez a 10 kbyte a szabad tárból foglal el igen nagy területet. A nagyfelbontású képernyőt használó programoknál tehát számolni kell jelentékeny tárcsökkenéssel. Ilyenkor a BASIC által használt RAM terület átszerveződik. A C 16-osnál a "BASIC vége" mutató lekerül $1800-ra, így a BASIC munkaterület mindössze 2 kbyte ($1000–$1800). A Plus/4-esnél a "BASIC eleje" mutató kerül fel $4000-ra, a bittérkép fölé. A $1000–$1800-ig terjedő 2 kbyte-os terület itt kihasználatlan, csak $4000–$FD00 területet használják a BASIC programok.

Bár a képpontokat külön-külön lehet ki- vagy bekapcsolni, a ki- és bekapcsolt pontok színét csak karakterhelyenként lehet megadni. Az első karakterhelyen (a

bittérkép első 8 byte-ja) lévő bekapcsolt (1) bitek a színmemória ide tartozó ($1C00) helyének alsó 4 bitjéből veszik a színinformációt, míg a fényerőt a fényerőmemória megfelelő helyének ($1800) 4.–6. bitjei határozzák meg. A kikapcsolt bitek (0) színe a színmemória 4.–7. bitjeiben vannak megadva, fényerejét pedig a fényerőmemória 0.–2. bitjei határozzák meg.

Áttérés normál bittérképes üzemmódra

A TED $FF06 címen lévő regiszter 5. bitjének magasra állításával térhetünk át bittérképes üzemmódra. Előtte azonban a $FF12 cím 3.–5. bitjeiben kell megadni, hogy a bittérképünk hányadik 8 K-ban kezdődik. A fényerőmemória kezdőcímét a $FF14-es TED regiszterben a 7.–3. biten kell megadni. Ezen kívül elengedhetetlen, hogy a $FF12 2. bitjét 0-ra állítsuk, ellenkező esetben meglepő módon a ROM területet tekinti a TED bittérképnek.

3–5. bit	Bittérkép	3–5. bit	Bittérkép
000	0000–1FFF	100	8000–9FFF
001	2000–3FFF	101	A000–BFFF
010	4000–5FFF	110	C000–DFFF
011	6000–7FFF	111	E000–FFFF

A következő programrészlet a bittérképes üzemmódra való áttérést mutatja be.

```
. 1100  AD  06  FF  LDA  $FF06   ;bittérképes üzemmód
. 1103  09  20      ORA  #$20    ;bekapcsolása
. 1105  8D  06  FF  STA  $FF06
. 1108  AD  12  FF  LDA  $FF12   ;előző bittérképhely és
. 110B  29  03      AND  #$03    ;ROM kapcsoló nullázás
. 110D  09  08      ORA  #$08    ;új bittérképhely =$2000
. 110F  8D  12  FF  STA  $FF12   ;beírás
. 1112  A9  18      LDA  #$18    ;szín- és fényerőmemória
. 1114  8D  14  FF  STA  $FF14   ;$1800-tól
```

2.2.5. TÖBBSZÍNŰ BITTÉRKÉPES ÜZEMMÓD

Ez az az üzemmód, amely alkalmas sokszínű képek megjelenítésére. A képernyő 160*200 képpontját egymástól függetlenül tudjuk az előre kiválasztott négy szín valamelyikére színezni, sőt ebből a négy színből kettő karakterhelyenként más és más is lehet.

A bittérkép elrendezése hasonló a normál bittérképes üzemmódnál leírthoz, de itt két szomszédos bit határozza meg egy képpont színét. Ennek megfelelően természetesen két pont helyett csak egy (elnyújtottabb) fog megjelenni, tehát a vízszintes felbontás a felére csökken.

A bitpárok állapota, és a hozzájuk tartozó képernyőpont színe között a következő az összefüggés:

Bit	Színforrás	Fényerőforrás
00	$FF15 0.–3. bitek	$FF15 4.–6. bitek
01	a karakterhely színmemóriájának 4–7. bitjei	a karakterhely fényerőmemóriájának 0–2. bitjei
10	a karakterhely színmemóriájának 0–3. bitjei	a karakterhely fényerőmemóriájának 4–6. bitjei
11	$FF16 0–3. bitek	$FF16 4–6. bitek

Áttérés a többszínű bittérképes üzemmódra

A többszínű bittérképes üzemmódra való áttérést kezdjük először a bittérképes üzemmódra való áttéréssel: ez a $FF06, $FF12, $FF14 regiszterek átírását jelenti, az előző részben ismertetett módon. Ezután a $FF07 4. bitjének 1-re állításával átkapcsolunk a többszínű üzemmódra.

2.2.6. EGYÉB GRAFIKÁVAL KAPCSOLATOS INFORMÁCIÓK

Az osztott képernyőkről eddig még nem esett szó. Ennek az az oka, hogy BASIC-ből való programozáskor is lényegében a megszakító rutin megfelelő pillanatban történő üzemmódváltásai teszik lehetővé, hogy különféle üzemmódok egy képernyőn egyszerre lehessenek jelen. Az osztott képernyő használatához tehát a megszakításokról szóló fejezet áttanulmányozására van szükség. Ezután azonban a grafikai üzemmódokat tetszés szerint kombinálhatjuk a képernyő bármely részén.

A megszakítási rendszer megismerése nélkül is van egy lehetőség osztott képernyő használatára. A grafikus üzemmódokat ugyanis a 0. lapon, a $83-as címen is lehet állítani. Ezt a címet a megszakító rutin figyeli és ennek alapján végzi el az üzemmód átkapcsolását. Ez a megoldás azonban nem állítja a $FF12 címen a bittérkép helyét. A $83-as címen megadható grafikus üzemmódok értékei a következők:

$00 – nincs kapcsolás
$60 – normál osztott képernyő
$A0 – többszínü bittérképes üzemmód
$E0 – többszinű osztott képernyő

Ha ezt a megoldást választjuk, akkor a fényerő memória kezdetét (a cím felső byte-ja) a $07FB címen állíthatjuk be. Ha a karakteres sávban saját (RAM) karakterkészlettel akarunk dolgozni, akkor a $07FA címre 0-t kell írni.

Lehetőség van bármely üzemmódban a teljes képernyő elmozdítására egy raszterpontnyival függőleges és vízszintes irányban is (finom scroll). Ehhez a TED $FF06-

os regiszterének 0.–2. bitjét (függőleges) vagy a $FF07-es regiszter 0.–2. bitjét (vízszintes) kell állítani. A görgetés használatakor a szebb grafikai hatás érdekében a görgetés irányába eső képernyőméretet célszerű lecsökkenteni. A függőleges méret a $FF06-os regiszter 4. bitjének 0-ra állításával 24 karaktersorra csökken. A képernyő szélessége a $FF07 3. bitjének 0-ra állításával 38 karakteresre állítható.

2.3. Számábrázolás, aritmetika

A BASIC-ben háromféle változótípust használunk. Szöveges (karakteres), egész, és valós típusúakat. Ebben a fejezetben az utóbbi két típussal foglalkozunk. A szöveges változók kezelése gépi programoknál sem jelent különösebb nehézséget.

2.3.1. EGÉSZ TÍPUSÚ VÁLTOZÓ

Egy gépi programban használhatunk 1, 2, sőt akárhány byte-os egész típusú változót annak megfelelően, hogy milyen számításokra van szükség. Részletesebben csak a 2 byte-os (16 bites) ábrázolással érdemes foglalkozni, mert BASIC-ben az egész típusú változók ilyenek, és ehhez hasonló az ennél hosszabb számok kódolása is.

Azok a programok, amelyeknek nem funkciója a számolás, meglepően ritkán használnak 1 byte-nál hosszabb számokat. Ha csak pozitív egészeket használunk; ezek 0-tól 255-ig terjedhetnek, de ha negatív értékekkel is akarunk számolni, akkor a számoknak −128 és +127 közé kell esniük. Az ilyen számokkal nagyon egyszerű műveletet végezni, mert az ennél hosszabbakkal ellentétben elférnek az akkumulátorban. Ez magyarázza azt, hogy a programozók törekednek úgy szervezni programjaikat, hogy ne kelljen hosszabb számokkal dolgozni.

Mivel a BASIC számoláscentrikus, az egész típusú változók itt 16 bitesek. Az ábrázolásmód előjeles, tehát negatív értékeket is megenged. A negatív számok kódolása a kettes komplemens szabályai szerint történik. Ezt már az 1.4-es alfejezetben részletesen ismertettük, azonban most 16 bitre kell ezt alkalmazni.

Pozitív szám értelemszerűen használható de a legmagasabb helyi értékű 15. bitnek 0-nak kell maradnia. Egy szám negáltját pedig úgy képezzük, hogy az összes bitet az ellentettjére váltjuk, utána az egész 16 bites számhoz 1-et adunk. Ebből következik, hogy a 15. bit lesz az előjel: 0 = pozitív, 1 = negatív.

Az alábbi program az $1005–$1006 címen elhelyezett szám ellentettjét képezi. Az eredmény az eredeti helyen keletkezik.

```
LDA   $1005   ;felső byte
EOR   #$FF    ;ellentett
STA   $1005   ;tárolás
LDA   $1006   ;alsó byte
EOR   #$FF    ;ellentett
```

```
CLC             ;C=0
ADC   #$01      ;+1
STA   $1006     ;tárolás
LDA   $1005     ;felső byte
ADC   #$00      ;+maradék: A=A+C
STA   $1005     ;tárolás
```

Lássunk néhány példát 16 bites egész számok (BASIC-ben %-jel jelzi) kódolására.

Decimális	Bináris	Hexa	
0	00000000 00000000	00 00	
+1	00000000 00000001	00 01	
+2	00000000 00000010	00 02	
+10	00000000 00001010	00 0A	
+32767	01111111 11111111	7F FF	(legnagyobb)
–1	11111111 11111111	FF FF	
–2	11111111 11111110	FF FE	
–10	11111111 11110110	FF F6	
–32768	10000000 00000000	80 00	(legkisebb)

A számábrázolással végezhetünk néhány kísérletet. Nyomjuk be a RESET gombot (STOP nélkül), majd írjuk be BASIC-ben:

A% = 1

Nézzük meg monitorból a $1005–$1006-os címek tartalmát:

>1005

A visszajelzés:

>1005 00 01 ... stb.

A fenti táblázatban megnézhetjük, hogy ez a 00 01 felel meg a +1-es értéknek. Felül is írhatjuk például a –10-nek megfelelő FF F6-tal. Ha visszatértünk BASIC-be, írjuk be:

PRINT A%

A visszajelzés a vártnak megfelelően –10 lesz. (A 2.8. alfejezetben van leírva, hogy hogyan lehet egy BASIC változót a tárban megtalálni. Addig elég azt tudni, hogy a RESET lenyomása után az első változó tartalma a $1005-ös helyre kerül.)

Az egy- és a kétbyte-os számok összeadására az 1.4. alfejezetben több példát is láthattunk. Az alábbi példa az előbbi A%-hoz hozzáad –10-et (levon 10-et).

```
CLC             ;C = 0
LDA   $1006     ;A% alsó byte
```

```
ADC   #$F6    ;-10 alsó
STA   $1006   ;A% alsó
LDA   $1005   ;A% felső
ADC   #$FF    ;-10 felső
STA   $1005   ;A% felső
BRK
```

A módszer a kivonásnál hasonló.

Miden összeadás és kivonás után a V-bit jelzi a belső túlcsordulást, tehát a kettes komplemens szerinti határok túllépését. 8 bites számnál pl. 64 + 64 (dec.) már túlcsordulást eredményez, mert az eredmény (128) már negatív számot jelent. Az előbb leírt rutin legvégén kell a V-bitet vizsgálni. Egy BVS utasítással a hibát lekezelő rutinra lehetne ugrani.

A BASIC ROM-ban több olyan rutin is van, amelyik egész típusú változókkal számol. Nagy hiányosságuk azonban, hogy az itt leírt rutinnál lényegesen lassúbbak. Ennek az okát később részletezzük.

2.3.2. VALÓS TÍPUSÚ VÁLTOZÓ

Ezt a típust az jellemzi, hogy értékei széles tartományban mozoghatnak pozitív és negatív irányban is, egész- és törtrészük is lehet. BASIC-ben ez az általánosan használt változó, azért is, mert az egész típusú változókkal az interpreter – meglepő módon – lassabban számol.

A valós típusú változók ábrázolása ún. lebegőpontos módon történik. Az egész számoknál a tizedespont (tizedesvessző) helye rögzített, a szám végén van, ezért fixpontos ábrázolásnak mondható. A lebegőpontos név onnan származik, hogy itt nincs előre rögzítve, hogy a szám hány egész, és hány tizedesjegyet tartalmaz. Tulajdonképpen csak az értékes jegyek száma rögzített, a tizedespont helye pedig adott határok között mozoghat.

A lebegőpontos ábrázolás mindig a számok normál alakjából indul ki. Tízes számrendszernél ez azt jelenti, hogy minden szám felbontható két szám szorzatára úgy, hogy az első szám 1 és 10 közé essen, a második pedig 10 valamilyen pozitív vagy negatív hatványa legyen. Pl. 1200-nál az első szám (amit mantisszának nevezünk) 1.2, ezt 1000-rel kell szorozni, hogy 1200-at kapjunk. Az 1000 a 10-nek a 3. hatványa, a kitevő (exponens) tehát +3.

A mantissza és az exponens minden számot egyértelműen meghatároz. A mantissza számjegyeinek száma dönti el a szám relatív pontosságát, tehát azt, hogy hány értékes jegye lesz. A kitevő értékének határai az átfogott számtartományt határozzák meg.

Szám	Mantissza	Kitevő
1.0	+1.0	0
10.0	+1.0	+1
73.0	+7.3	+1
−120.0	−1.2	+2
0.1	+1.0	−1
−0.019	−1.9	−2

Kettes számrendszerben a helyzet hasonló. Itt olyan két szám szorzatára bontjuk fel a számot, ahol az első 1 és 2 közé esik, a második pedig 2 valamilyen pozitív vagy negatív hatványa.

Pl. a %1010 (10) számnál a mantissza 1.01 (1.25), és ezt %1000-val (8) kell szorozni, hogy az eredeti %1010-as számot kapjuk. Ez 2-nek a %11. (3.) hatványa, tehát a kitevő %11.

Szám	Mantissza	Kitevő (bin.)
1.0	+1.0	0
−10.0	−1.0	+1
101.0	+1.01	+10
−1001.0	−1.001	+11
0.1	+1.0	−1
−0.011	−1.1	−10

Megfigyelhető, hogy a mantissza mindig "1."-tal kezdődik. Nem véletlenül, hiszen 1 és 2 közé kell esnie. Ezt a gép a későbbiekben leírt módon kihasználja.

A Commodore lebegőpontos számábrázolása a következőek szerint történik.

A kitevőt 1 byte-on, a mantisszát 4 byte-on tárolja, tehát egy szám 5 byte-ot foglal le. A kitevőt először kettes komplemens formájúra alakítja, majd az így kapott értékhez még $81-et (129-et) hozzáad (offset). Így pl. a 0 kitevőt a $81 (129) kódolja. A megengedett kitevőértékek −128 és +126 között lehetnek, amit a kitevőbyte-ban $01–$FF kódol. A $00 jelenti azt, hogy maga a szám 0. (függetlenül a mantisszától)

Kitevő érték	Kódolt érték
$00 (0)	$81
+$01 (1)	$82
+$02 (2)	$83
+$7E (126)	$FF
−$01 (−1)	$80
−$02 (−2)	$7F
−$80 (−128)	$01

A mantissza tárolása a kitevőt követő 4 byte-on történik, a magasabb helyi értéktől az alacsonyabb felé haladva. A "kettedes" pontot balról az első és második bit között kell elképzelni. Mivel a mantissza első bitje mindig 1, ezért nem jelent információveszteséget, ha nem tároljuk (de azért mindig oda kell képzelni). A

helyén, tehát az első biten ábrázolhatjuk a mantissza előjelét. Itt is a 0 jelenti a pozitív, az 1 pedig a negatív előjelet. Azonban ez más, mint az egész számoknál a kettes komplemensnél, mert itt egy pozitív szám csak az előjelbitben különbözik az ellentettjétől. A mantisszánál tehát nincs komplemens kódolás.

Aki bonyolultabb számolásokat végez, annak fontos lehet, hogy hány értékes jeggyel számol a gép. Az már kiderült, hogy a mantissza 32 bites, tehát kettes számrendszerben értelmezve 32 értékes jeggyel számolunk. Hogy ez hány tizes számrendszerbeli jegynek felel meg, az kiderül, ha kiadjuk a következő BASIC parancsot:

```
PRINT 32 * LOG(2) / LOG(10)
```

Az eredmény: 9.63295986

Ez talán úgy értelmezhető, hogy pontosabb, mintha 9 tízes számrendszerbeli jeggyel számolna, de pontatlanabb, mintha 10-zel. A PRINT mindenesetre maximum 9 jegyet ír ki, az előbb leírt példa is ezt mutatja.

Lássunk néhány példát lebegőpontos számokra:

Decimális érték	Bináris érték	Kitevő	Mantissza
1.0	+1.0	81	00 00 00 00
2.0	+10.0	82	00 00 00 00
−2.0	−10.0	82	80 00 00 00
4.5	+100.1	83	10 00 00 00
−4.5	−100.1	83	90 00 00 00
0.75	+0.11	80	40 00 00 00
1.70141183E+38		FF	7F FF FF FF
2.93873588E−39		01	00 00 00 00
0	0	00	00 00 00 00

Az egész típusú változókhoz hasonlóan itt is végezhetünk néhány kísérletet: A RESET benyomása után írjuk be:

```
A = 1
```

Majd monitorból kérdezzük le a $1005-től kezdődő tártartományt:

```
>1005
```

A visszajelzés:

```
>1005 81 00 00 00 00 ... stb.
```

Amint várható volt, ez az 1 lebegőpontos formában. Javítsuk ki a kitevőt 2-re, ezt a $83 kódolja (a kitevőtáblázatban megnézhetjük):

```
>1005 83
```

Kérdezzük le BASIC-ből az A értékét:

PRINT A

Az eredmény 4 lesz.

Ugyanígy könnyen módosítható az előjel és a mantissza is.

Ha törtszámokkal dolgozunk, nem szabad elfelejteni, hogy kettes számrendszerben kifejezve az egytized is végtelen szakaszos "kettedes" tört, akárcsak pl. az 1/7 tizedes törtben kifejezve. Míg a tizedes törteknél ez a ritkább, kettes számrendszerben nagyon gyakori.

Egytizednél a mantissza:

1.100110011001100110011001100...

Ábrázolása mégis a következő:

7D 4C CC CC CD

Az 1100 szakaszok periodicitása megszakad az utolsó bitnél. Ez annak a következménye, hogy a gép nem egyszerűen levágja azt a részt, ami nem fér a 32 bitbe, hanem kerekít a túlcsorduló részt figyelembe véve. Mivel a 33., tehát a túlcsorduló bit 1-es értékű, felfelé kerekít, azaz 1-et hozzáad a bennmaradó részhez. Ha a túlcsorduló bit 0, akkor a 32. bit utáni rész egyszerűen elmarad. Ez ahhoz hasonlít, ahogy a tizes számrendszerben

$$1 / 3 = 0.333333333$$
$$2 / 3 = 0.666666667$$

módon kerekítjük a végtelen tizedes törteket.

Lebegőpontos számokkal műveletek végzése általában nem egyszerű, ezekre inkább a BASIC ROM rutinjait érdemes használni.

2.3.3. ARITMETIKAI RUTINOK

A BASIC ROM-ban több olyan rutin van, amely egy vagy két lebegőpontos számmal aritmetikai vagy logikai műveletet végez el. Ide tartozik a számok szöveges formájúvá alakítása, és a szövegesen leírt szám egész, vagy lebegőpontos ábrázolásúvá alakítása is. Ezeket a rutinokat a továbbiakban egyszerűen aritmetikai rutinoknak fogjuk nevezni.

Az aritmetikai rutinok úgy működnek, hogy az operandusaiknak mindig ugyanazon az előre rögzített helyen kell lenniük. Ezeket a tárterületeket, ahol az operandusokat tároljuk, lebegőpontos (floating point) akkumulátoroknak nevezzük, röv.: FAC. Két ilyen FAC van, mivel a műveletek egy- vagy kétoperandusúak. Megkülönböztetésül saját nevük is van: a $61–$66 területen lévőt nevezzük csak FAC-nak, a $69–$6E területet lefoglalót pedig ARG-nak (argumentum).

Tárcím		Tartalom
$61	FAC	exponens
$62		mantissza
$63		mantissza
$64		mantissza
$65		mantissza
$66		előjel
$69	ARG	exponens
$6A		mantissza
$6B		mantissza
$6C		mantissza
$6D		mantissza
$6E		előjel

$6F az előjel-összehasonlítás eredménye
00 = egyező, FF = különböző

Nagyon fontos megjegyezni, hogy a BASIC változók tárolásával ellentétben itt az előjel számára külön hely van fenntartva, a mantissza előjelbitje helyén a valódi érték (mindig 1) van! Az előjelbyte-nak csak a 7. bitje számít: 0 = pozitív, 1 = negatív, de a negatív értéket az FF jelöli, a pozitívat a 0.

1 = 81 80 00 00 00 00
–1 = 81 80 00 00 00 FF

A rutinok hibátlan működése általában azt is megkívánja, hogy a $6F-be be legyen írva az előjel-összehasonlítás eredménye. Gyakran az okoz hibát, hogy a rutin hívásakor a Z-bit értéke 1 volt.

Az első csoportba az alapműveletek tartoznak:

összeadás :	FAC = ARG + FAC	$9E9E
kivonás :	FAC = ARG – FAC	$9E87
szorzás :	FAC = ARG* FAC	$A07B
osztás :	FAC = ARG / FAC	$A197
hatványozás:	FAC = ARG ↑ FAC	$A5EE

Célszerű úgy meghívni őket, hogy előtte a FAC kitevőjét az akkumulátorba töltjük. Ekkor a hibakezelés is megtörténik (pl. ARG / FAC), és a futás is gyorsul (ARG + FAC).

A rutinok egy másik csoportja a tár egy előre megadott részén elhelyezett lebegőpontos számot az ARG-ba másolja, beírja az előjelbyte értékét $6E-be, és elvégzi az előjelösszehasonlítást is. Visszatérés előtt beolvassa az akkumulátorba FAC kitevőjét.

A beolvasandó szám kezdőcímének A/Y-ban kell lennie: az akkumulátorban az alsó, az Y-regiszterben a felső byte-nak.

beolvasás ROM-ból: ARG = (A/Y) $A0DC

beolvasás RAM-ból: ARG = (A/Y) $A107

Az alábbi két rutin a FAC-ba olvas be. Az előjelösszehasonlítás itt elmarad.

beolvasás ROM-ból: FAC = (A/Y) $A221

beolvasás RAM-ból: FAC = (A/Y) $A21F

Az előbbi négy beolvasórutin hívása után a BEQ utasítás jelentése: ugrás, ha FAC = 0. Az alapműveletek rutinjai ezt kihasználják.

A ROM-ból való olvasás értelmetlennek tűnhet, de tudni kell, hogy sok matematikai állandót találhatunk itt lebegőpontos formában kódolva.

Az egyik legnehezebb feladat a lebegőpontos szám karakteres (ASCII) formájúvá konvertálása. Ugyanilyen bonyolult az ellenkezője is, tehát a karakteresen leírt szám lebegőpontossá alakítása. A két rutin:

konverzió: FAC → ASCII $A46F

konverzió: ASCII → FAC $A37F

Az első rutin használata a könnyebb. A FAC értékének megfelelő szám szöveges formában a $0100-tól kezdődő tárterületre kerül. A FAC értéke a végrehajtás közben elvész.

pl.: 85 80 00 00 00 FF = –16

A FAC-ba beírás után futtassuk le a $A46F-es rutint:

```
A 2000   JSR   $A46F
         BRK
G 2000
```

Az eredmény: >0100 2D 31 36 00 .. .. : –16...

A másik rutin használata már sokkal nehezebb, mert használja a $0473-as (CHRGET) rutint. Erről a rutinról később még lesz szó. Most csak azt mutatjuk meg, hogy az ASCII→ FAC konverzió a gyakorlatban hogyan valósítható meg. Legyen a feladat az, hogy az előző példában szereplő –16-ot akarjuk a FAC-ba, lebegőpontos formába konvertálni.

```
LDA   #$00    ;szövegmutató: $0100-ra
STA   $3B     ;alsó
LDA   #$01    ;szövegmutató
STA   $3C     ;felső byte
JSR   $0479   ;CHRGOT (első jegy)
JSR   $A37F   ;konverzió
BRK
```

A szám szöveges formában egyébként a RAM bármely területén lehet. Egyetlen kikötés, hogy a végét egy 0 tartalmú rekesz jelezze.

Egy BASIC programot megzavar az, hogy ha futás közben a $3B–$3C értékét módosítjuk. Ha ezt el akarjuk kerülni, írjuk vissza a módosítás előtti értékét a konverzió után.

Az egész–lebegőpontos, és a lebegőpontos–egész konverziót gyakran kell használni többek közt azért is, mert a legtöbb rutin lebegőpontos számokkal dolgozik.

PRINT A%

végrehajtása során az interpreter először lebegőpontossá alakítja az A% egész számot, majd konvertálja szöveges formájúvá ($A46F), és ezt írja ki a képernyőre. Nagy hiányosság, hogy nincs fixpontos szorzás, osztás, hatványozás, és ASCII konverzió. Emiatt a lebegőpontossá alakítás az egyetlen járható út egész számok használata esetén.

Lebegőpontos számnak egésszé alakítását a $A327-es rutin végzi. Az eredmény 4 byte-os és a $62–$65-ös tartományba, tehát a mantissza helyére kerül. A negatív számok kettes komplemens kódolásúak. Aki csak kétbyte-os számokkal akar számolni, annak csak a $64–$65-ös címeket kell olvasnia. Ez a rutin tulajdonképpen egyformán használható 1–4 byte-os egészekkel való számoláskor.

Segítségünkre lehet ekkor a mantissza invertáló rutin is, mellyel itt az előbbi egész szám negáltját (ellentettjét) képezhetjük. Használatakor arra kell ügyelnünk, hogy a kerekítő érték ($70) nulla legyen.

Az átalakító rutinok:

mantissza invertálás: $9F7B (előjellel)
mantissza invertálás: $9F81 (előjel nélkül)

4 byte-os pozitív egész → FAC $9F0B
4 byte-os előjeles egész → FAC $9F06

Az egész számok lebegőpontossá alakítása igényel bizonyos előkészítést. Először is az átalakítandó számot be kell írni a mantissza helyére, a $62–$65 területre ($62 a legmagasabb helyi értékű). A kitevő ($61) helyére $A0-t kell írni, és 0-t a $66-ba (előjel) és a $70-be (kerekítés). Ezután a $9F0B-s rutin meghívható, ha csak pozitív egészet akarunk konvertálni. A $9F06 előtt ezen kívül még az alábbi utasításokat is:

```
LDA   $62
EOR   #$FF
ROL
JSR   $9F06
```

vagyis az előjel negáltját a C-be kell juttatni.

2.3.4. FÜGGVÉNYEK KIÉRTÉKELÉSE

A BASIC függvények egy részének értéke egyszerűen számolható, részletesen ezekkel nem érdemes foglalkozni. Egy másik csoport azonban okozhat problémát. Ide tartoznak pl. a SIN, COS, LOG függvények. A függvényértékek itt túlnyomórészt irracionális számok. Kiszámításukra nincs olyan módszer, hogy a négy alapművelettel végzett véges számú lépéssel megkaphatnánk a függvényértéket. Ezt a csoportot nevezzük transzcendens függvényeknek.

A transzcendens függvények értékét a Taylor-polinomjukkal közelíthetjük. A közelítés szó itt egy kicsit félrevezető. Matematikai értelemben a pontos érték kiszámítása ennek a polinomnak végtelen sok tagjával lehetséges, de ha figyelembe vesszük azt, hogy a gép úgyis csak 32-bites pontossággal számol, az ehhez képest pontos érték kiszámításához már a polinom első néhány tagja elegendő.

A számítási módszer tehát a következő:

$$Y = A0 + A1 * X + A2 * X\uparrow 2 + A3 * X\uparrow 3 + A4 * X\uparrow 4 \ldots \text{ stb.}$$

A polinomszámító rutin a $A6C9 címen indul. Indítás előtt az ARG-ba kell tölteni a függvény független változóját (X-et). Az akkumulátorba és az Y-regiszterbe be kell tölteni a polinomkonstansok táblázatának a kezdőcímét. Egy ilyen táblázatnak a következőképp kell kinéznie:

1 byte : fokszám
5 byte : a legmagasabb fokszámú tag együtthatója
:
5 byte : a nulladfokú tag együtthatója (konstans)

Az n-edfokú polinomnál:
fokszám + (n + 1) együttható = (n + 1) * 5 + 1 byte

A számítás nem a fent leírt módon, hanem a Horner-elrendezés szerint történik: pl. 4-ed foknál:

$$Y = (((A4 * X + A3) * X + A2) * X + A1) * + A0$$

Ezzel a módszerrel a negyedfokú polinom négy szorzással és négy összeadással számítható.

A polinomszámító rutint felhasználva a legkülönfélébb transzcendens függvényeket hozhatjuk létre, és BASIC USR függvény felhasználásával BASIC bővítésként is alkalmazhatjuk ezeket.

2.4. A megszakítások programozása

A megszakítással kapcsolatos programozási technika nem tartozik a leggyakrabban használt programozási fogások közé, pedig az egyik leghatékonyabb eszköz és a leglátványosabb megoldásokat csak ezzel lehet megvalósítani. A BASIC utasításokban is csak rejtve szerepel és így is csak ritkán. Például a SOUND utasítás végrehajtásakor a hang még szólhat, amikor a gép már a következő utasítást (utasításokat) is végrehajtotta. A SOUND csak bekapcsolja a hangot, és egy időzítőt állít be. Úgy tűnik, mintha ez az időzítő párhuzamosan működne a főprogrammal. Valójában azonban ezt is a megszakító rutin kezeli. Minden olyan esetben, amikor a főprogramunkkal párhuzamosan futó programot szeretnénk működtetni, a megszakítási technikát kell alkalmazni. Ezt a következőkben két szinten fogjuk tárgyalni.

Akik nem tudnak, vagy nem akarnak a megszakítások bonyolult működési mechanizmusában elmélyedni, de mégis szeretnék kihasználni a benne rejlő óriási lehetőségeket, azoknak az első részben tárgyalt megszakító rutin-kiegészítési lehetőségeket célszerű átolvasni.

A második részben a megszakítások teljes működési mechanizmusa ismertetésre kerül. Ez a rész elsősorban azoknak ajánlható, akik az eredetitől független saját megszakító rutint akarnak használni. Ilyenre gyakran van szükség például játékprogramoknál vagy különleges grafikai hatások programozásánál.

2.4.1. A MEGSZAKÍTÓ RUTIN KIEGÉSZÍTÉSE

A megszakításnak leegyszerűsítve az a szerepe, hogy az éppen futó gépi programtól függetlenül, vele párhuzamosan ellásson bizonyos feladatokat. Mivel a gépben csak egy processzor van, benne egyszerre csak egy program futhat, így ezt a feladatot csak úgy láthatja el, hogy bizonyos események bekövetkeztekor (pl. szabályos időközönként) felfüggeszti a főprogram futását, és végrehajtja a kívánt megszakító rutint. Amikor befejezte, onnan folytatja a futást, ahol abbahagyta. A megszakítás úgy van megszervezve, hogy semmilyen főprogram futását ne zavarhassa meg. Ez azt is jelenti, hogy az összes regiszter ugyanolyan értékű a megszakításból való visszatéréskor, mint előtte.

Az ezután következők a gép bekapcsolás utáni állapotára vonatkoznak, tehát amikor a megszakítások vezérlésébe még nem avatkoztunk bele.

A megszakító rutin másodpercenként 100-szor hajtódik végre, melyet a raszterszámláló vált ki adott raszterértékeknél. Egy képernyő kirajzolásának ideje alatt (kb. 1/50 másodperc) tehát két megszakítás történik.

A megszakító rutin a következő funkciókat látja el:

- Kiválasztja a rendszer ROM-ot. (Erről a memórialapozásnál lesz majd szó.)
- Osztott képernyőnél (karakteres + bittérkép) kiválasztja az aktuális raszterhelyzetnek megfelelő grafikus módot.

- Beállítja a következő megszakítás raszterszámláló értékét.

- A STOP billentyű lenyomottságát ellenőrzi (BASIC programok megszakításához), és a TI változó értékét növeli ($A3, $A4, $A5, 3 byte-os egész). Ezt minden második megszakítás aktualizálja, tehát másodpercenként 50-szer frissül fel az értéke. Mivel minden ötödik növelés 2-vel növeli, így tulajdonképpen a 60-ad másodperceket számolja.

- A billentyűk lenyomottságát ellenőrzi, és a pufferbe írja az ASCII-kódjukat.

- Visszalapozza a $FB által megadott ROM-ot. Ez esetünkben ismét a rendszer ROM.

A ROM-ok lapozásának az alapkiépítésű C 16-osnál soha, Plus/4-esnél csak ritkán van jelentősége.

Egy megszakítás a következő módon zajlik le:

1. A raszterszámláló adott értéket elérve megszakítást kér. Ha az I-bit értéke 1, akkor a megszakítás nem jöhet létre.

2. A verembe kerül az utasításszámláló, a feltételregiszter és az összes egyéb regiszter értéke.

3. JMP ($0314) utasítással a megszakító rutinra kerül a vezérlés. Alapállapotban a $0314–$0315 vektor a $CE0E címet tartalmazza. Lefut az itt leírt rutin.

4. Visszatöltődik a veremből az összes regiszter értéke.

5. Folytatódik az eredeti program.

A JMP ($0314) indirekt ugrás teremt egyszerű lehetőséget a megszakítással kapcsolatos manipulációkra. Amíg a teljes rutin működését nem ismerjük, addig mást nem tehetünk, mint ezt a meglévő rutint egészítjük ki a sajátunkkal a következők szerint.

Először természetesen meg kell írni a kiegészítő programot. Arra vigyázzunk, hogy mindenképpen JMP $CE0E-vel végződjön. Az is lebénítja a gépet, ha ez a rutin végtelen ciklust tartalmaz.

Ha a kiegészítéssel elkészültünk, akkor át kell írnunk a $0314-es indirekt címet a saját programunk kezdőcímére.

Példaképpen nézzünk egy nagyon egyszerű programot:

```
A 1100  LDA  #$01
        STA  $0C00
        JMP  $CE0E
```

A példaprogram a képernyő bal felső sarkába A betűt ír. Ezt onnan sem a képernyő görgetése, sem egyéb törlés nem tünteti el, mert másodpercenként 100-szor újra beíródik.

Ezt a programot úgy kelthetjük életre, ha a $0314 indirekt címre $1100-nak megfelelő értéket írunk.

Monitorból az átállítás nagyon egyszerű:

```
>0314 00 11
```

Programból ugyanez:

```
SEI
LDA   #$00    ;rutin kezdet
STA   $0314   ;alsó byte-ja
LDA   #$11    ;rutin kezdet
STA   $0315   ;felső byte-ja
CLI
```

A SEI-re azért van szükség, mert ha egy megszakítás a két STA között történik, akkor hibás helyre ugrik a JMP ($0314).

A következő megszakítás-kiegészítéssel elérhetjük, hogy a háttér- és a keretszín más és más legyen a képernyő különböző részein.

```
LDA   $FF15
EOR   #$33
STA   $FF15
STA   $FF19
JMP   $CE0E
```

A színváltás a képernyő 20. és 21. sora között és a 25. sor végén történik. A felső tartományban a háttér és a keret világos, az alsóban pedig mindkettő sötét (esetleg fordítva). Annak ellenére, hogy a gép működésében (programozhatóságában) semmi változás nem történt, sikerült egy elég jó grafikai hatást elérni.

Programteszteléskor vagy hibakereséskor segíthet az, ha bizonyos címek értékei folyamatosan követhetők a képernyőn. Az alábbi program a 0. lapos óra (TI) értékeit írja a képernyőmemóriába.

```
      LDX   #$02
CIKL  LDA   $A3,X
      STA   $0C00,X
      DEX
      BPL   CIKL
      JMP   $CE0E
```

A rutin elég egyszerű, de az időérték sajnos nem olvasható ki, mert a kijelzés nem karakteres formájú. A későbbiekben láthatunk példát a TI$ karakteres kiírására is.

Kiegészítő rutinunkkal a megszakítás pillanatában érvényes PC, AC, XR, YR, SR értékeket is lekérdezhetjük anélkül, hogy a főprogramot megzavarnánk.

```
TSX
LDA   $0101,X
```

az Y-regiszter értékét olvassa be.

A regiszterek értékének beolvasása:

```
0101,X   YR
0102,X   XR
0103,X   AC
0104,X   SR
0105,X   PC   alsó byte
0106,X   PC   felső byte
```

A megszakító rutin két részre oszlik, és ezek felváltva működnek. A nagyobbik rész végzi a legtöbb funkciót: billentyű-lekérdezés, óra aktivizálás (TI). A másik ágra elsősorban az osztott képernyő kezelése miatt van szükség. A fő ág kezdetére újabb indirekt cím mutat ($0312). Ennek átírásával másodpercenként 50-szer végrehajtott rutinnal lehet kiegészíteni a megszakítást. Ezeket a kiegészítéseket a JMP $CE42 utasítással kell befejezni.

A következő példaprogram a TI$ karakteres formában történő kiírását végzi.

```
         JSR   $9531
         STY   $5E
         DEY
         STY   $71
         LDY   #$06
         STY   $5D
         LDY   #$24
         JSR   $A4FA
         LDX   #$05
OLVAS    LDA   $00FF,X
         STA   $0C00,X
         DEX
         BPL   OLVAS
         JMP   $CE42
```

Működését a $0312–$0313 cím átírásával indíthatjuk el. Ez a rutin használja a FAC-ot és néhány BASIC vektort, emiatt BASIC programok működését megzavarja. Írhatunk olyan óraprogramot is, amelyik nem zavarja meg a BASIC program futását, de ez bizonyosan hosszabb és bonyolultabb lesz. Legcélszerűbb erre olyan új órát létrehozni, melyet a megszakítás karakteres formában aktivizál.

Játékprogramoknál szokásos megoldás, hogy a megszakító rutin folyamatosan (szabályos időközönként) átdefiniál bizonyos karaktereket. Így lehet forgó, villogó, lüktető, hullámzó karaktereket használni úgy, hogy a főprogramnak az átdefiniálással egyáltalán nem kell foglalkoznia. A változtatás sebességét úgy lehet lassítani, hogy nem minden megszakítás végez átdefiniálást. Ez egy számláló felhasználásával érhető el.

A megszakító rutin lerövidítése a kiegészítéshez hasonló módon történik. Erre már az első fejezetben is láthattunk példát. A BASIC STOP letiltása a $0312-es indirekt cím átírásával valósítható meg. A mutatónak $CE45-re való átírásával kimarad egy szubrutin végrehajtása. A többi funkció is kiiktatható hasonló módon (bár ennek kevés értelme van).

2.4.2. A MEGSZAKÍTÁSOK MŰKÖDÉSE

Megszakítást (interruptot) a következő eszközök válthatnak ki:

- raszterregiszter
- fénytoll (fényceruza)
- 1. időzítő
- 2. időzítő
- 3. időzítő

Együtt letilthatók a SEI utasítással, de az egyes megszakítások külön-külön is letilthatók vagy engedélyezhetők. Ez a TED $FF0A című regiszterének állításával lehetséges.

Bit sorszám	Eszköz
1.	raszter
2.	fénytoll
3.	1. időzítő
4.	2. időzítő
6.	3. időzítő

Az ily módon engedélyezett eszköz megszakítási kérelmet vált ki, amely csak akkor eredményez megszakítást, ha az I-bit 0 értékű. Ez esetben a processzor még befejezi az éppen végzett műveletet, majd az aktuális programszámláló (felső-alsó byte) és a feltételregiszter értékét a verembe tölti. Az I-kapcsoló értékét 1-re állítja, hogy újabb megszakítás ne következhessen be a rutin működése közben. A programszámlálóba betöltődik az az érték, amit a $FFFE–$FFFF címek tartalmaznak. A ROM-ban itt a $B3, $FC értékek vannak, tehát a megszakító rutin futása $FCB3 címen kezdődik.

A rutinból való visszatérésre az RTI utasítás ad lehetőséget. Hatására visszatöltődik a feltételregiszter és az utasításszámláló megszakítás előtti értéke, és az eredeti program futása a megszakítás helyétől folytatódik. Az I természetesen visszaáll 0-ra.

Érdemes megnézni, hogy mit találunk a $FCB3 címen:

```
. FCB3  48          PHA              ;AC
. FCB4  8A          TXA
. FCB5  48          PHA              ;XR
. FCB6  98          TYA
. FCB7  48          PHA              ;YR
. FCB8  8D  D0 FD   STA  $FDD0       ;rendszer ROM
. FCBB  4C  00 CE   JMP  $CE00       ;megszakító rutin
. FCBE  A6  FB      LDX  $FB         ;eredeti ROM BANK
. FCC0  9D  D0 FD   STA  $FDD0,x     ;visszaállítás
. FCC3  68          PLA              ;YR
. FCC4  A8          TAY
```

```
. FCC5   68         PLA              ;XR
. FCC6   AA         TAX
. FCC7   68         PLA              ;AC
. FCC8   40         RTI
```

A legelső lépés az, hogy a verembe kerül az akkumulátor, az X- és az Y-regiszter tartalma. Enélkül a főprogramot megzavarná minden megszakítás, hiszen automatikusan csak a feltételregiszter és az utasításszámláló értékét menti ki.

A rendszer ROM kiválasztásának csak akkor van jelentősége, ha éppen másik ROM-ban működő programot szakítottunk meg. A megszakító rutin $CE00-tól kezdődő fő része a rendszer ROM-ban van, ellentétben a $FCB3-assal, mert ez közös ROM-területen van. Ez a program JMP $FCBE-vel végződik. Itt az aktuális (eredeti) ROM kiválasztása és a regiszterértékek visszaállítása következik.

A $CE00-tól kezdődő rész:

```
. CE00   BA            TSX
. CE01   BD  04  01    LDA   $0104,X   ;feltételregiszter
. CE04   29  10        AND   #$10      ;B bit
. CE06   D0  03        BNE   $CE0B     ;nem 0
. CE08   6C  14  03    JMP   ($0314)   ;BREAK
. CE0B   6C  16  03    JMP   ($0316)   ;$CE0E
```

Látható, hogy a megszakító rutin már a legelején két ágra szakad attól függően, hogy a megszakítás pillanatában a B-bit 0 vagy 1 volt. Ha 0, vagyis ha a megszakítást nem a BRK utasítás idézte elő, akkor a

JMP ($0314)

hajtódik végre, tehát egy "normális" megszakítási ciklus, amelyik a $CE0E-nél kezdődik (amíg át nem írjuk).

Ha B = 1 volt, akkor a

JMP ($0316)

utasítást hajtja végre a gép, ami alapállapotban $F44C-re ugrást eredményez. Ez kiírja a BREAK üzenetet, monitor R parancsot hajt végre, és a gép ezután monitor üzemmódban marad.

Az mindenesetre látható, hogy a BRK nem csak erre használható, teljesen "szokványos" megszakításokat is kiválthatunk vele. Ez a megoldás bizonyos esetekben szükséges lehet, pl. időtakarékossági okokból.

Önálló megszakító rutin írásának egyszerűbb formája az, amikor a $FCB3–$FCC8 és a $CE00–$CE0B ROM-programokat még használjuk. Mindezek feltétlenül működnek, amíg a $FFFE–$FFFF címek tartalma $FCB3. Ennél a módszernél is a $0314-es indirekt címet kell átírni úgy, hogy a saját rutinunkra mutasson. Ennek azonban JMP $FCBE-vel vagy JMP $FCC3-mal kell végződnie. (Mivel ritkán írunk új megszakító rutint a bővítő ROM-ok programjaihoz, gyakorlatilag a JMP $FCC3 is mindig alkalmazható.)

Nézzünk példát egy minimális hosszúságú megszakító rutinra:

```
LDA   $FF09
STA   $FF09
JSR   $DB11   ;SCNKEY, leírása: 2.5. alfejezet
JMP   $FCC3   ;visszatérés
```

Ez a rutin nagyságrendekkel rövidebb, mint az eredeti, mégis még BASIC-ben is alig érezhető a különbség, gépi programból pedig egyáltalán nem.

A megszakítás rendszerének gyökeres átalakítása akkor válhat szükségessé, ha a főprogram futása közben a ROM egyáltalán nem lehet jelen (RESET letiltás).

Az itt közölt rész megértéséhez elengedhetetlen a 2.7. alfejezetben leírt memórialapozási technika alapos ismerete.

Ha a RAM-ot lapozzuk felülre, akkor a $FFFE–$FFFF RAM-beli értéke felülírható. Ide kell beírnunk az új megszakítási vektort, amely a saját rutinunkra mutat.

Ennek a rutinnak a következőképpen kell kinéznie:

```
PHA
TXA
PHA
TYA
PHA
...    (a végrehajtandó rutin)
PLA
TAY
PLA
TAX
PLA
RTI
```

A regiszterek kimentési sorrendje lehet más is. A kipontozott rész helyére kell a meszakító rutint tenni. Ez lehet például az előbb leírt "minimális" rutin is:

```
LDA   $FF09
STA   $FF09
STA   $FF3E   ;ROM választás
JSR   $DB11   ;SCNKEY
STA   $FF3F   ;RAM választás
```

Itt az SCNKEY a ROM-ban fut, de futhatna RAM-ban is, ehhez egyszerűen át kell másolni. A fenti rutin csak $8000-es cím alatt működhet a lapozás miatt.

2.4.3. A RASZTERMEGSZAKÍTÁS HASZNÁLATA

A raszterszámláló tulajdonképpen azt számolja, hogy a képernyőkép hányadik sorának kiírása van folyamatban. (Tudni kell, hogy a TV vagy monitor képernyőjére a kép soronként íródik ki, másodpercenként 50-szer) Ha azt akarjuk, hogy az általunk előidézett (képernyőre vonatkozó) változtatás csak egy adott képernyősávra vonatkozzon, akkor a sorszámlálótól függően kell a változtatást előidézni és eltüntetni.

A sorszámláló a TED-ben a $FF1D címen található, a legfelső bitje pedig a $FF1C 0. bitjén. Értéke 0-tól $137 (311)-ig nő, majd újra 0-tól kezdi a számolást. Az egy soron belüli pozíció számlálója a $FF1E címen van. Értéke 0-tól $E4 (228)-ig változik. Mivel az értéke a legrövidebb programok futási idejéhez képest is túl gyorsan változik, jelentősége kevés.

A különleges grafikus hatások eléréséhez a raszterszámláló programból történő figyelése nem megoldás, mert a figyelés mellett már semmi más nem futhat. Kiváló lehetőség viszont a raszterszámláló által kiváltott megszakítás kihasználása. Erre jó példa a BASIC-ből megismert osztott képernyő használata. Ezt ott a megszakítás úgy kezeli, hogy amíg a képernyő felső 20 karaktersorának kiírása folyik, addig bittérképes üzemmódot kapcsol. A 20. és 21. sor között kiváltott megszakítás átvált normál karakteres üzemmódra, a 25. sor után érkező megszakítás visszavált grafikus üzemmódra.

A következő megszakítás bekövetkezésének rasztersorát a TED $FF0B (a legfelső bit a $FF0A 0. bitje) című regiszterébe kell írni. A beírt értékek jelentéséhez segítséget nyújt a következő táblázat:

Raszter-számláló		Jelentés	
4	$04	képernyőablak kezdete	(25 * 40 karakter)
204	$CC	képernyőablak vége	(25 * 40 karakter)
8	$08	képernyőablak kezdete	(24 * 38 karakter)
200	$C8	képernyőablak vége	(24 * 38 karakter)
283	$11B	felső keret kezdet	(311 után 0)
244	$F4	alsó keret vége	
245– –282	$F5– –$11A	nem látható	

Ha a $FF0A–$FF0B-be 311-nél nagyobb számot írunk, nem vált ki megszakítást.

A következő példa egy olyan új megszakító rutin, amelyik minden 16. rasztersorban generálódik, és változtatja a keret és a háttér színét.

```
LDA   $FF09
STA   $FF09
LDA   $40
```

```
STA   $FF15   ;háttér
STA   $FF19   ;keret
CLC
ADC   #$10
STA   $40
STA   $FF0B   ;következő megszakítás
JSR   $DB11   ;billentyűzet (SCNKEY)
JMP   $FCC3   ;vissza megszakításból
```

Erre az új rutinra, ha például $2000-nél kezdődik, a következő programmal tudunk átkapcsolni:

```
SEI
LDA   $00     ;kezdőcím alsó
STA   $0314
LDA   #$20    ;kezdőcím felső
STA   $0315
LDA   $FF0A   ;raszterszámláló felső bit
AND   #$FE    ;nullázás
STA   $FF0A
LDA   #$05    ;zöld
STA   $40
JMP   $FF45   ;(MONITOR)
```

2.4.4. AZ IDŐZÍTŐK HASZNÁLATA

A TED-ben 3 darab kétbyte-os időzítő (számláló) is van, amelyek megszakítást is képesek kiváltani. Elhelyezkedésük a következő:

```
1. időzítő   $FF00   :alsó
             $FF01   :felső

2. időzítő   $FF02   :alsó
             $FF03   :felső

3. időzítő   $FF04   :alsó
             $FF05   :felső
```

Értékük egyesével csökken 884 kHz frekvenciával. Ez az érték bármikor kiolvasható LDA-val, és STA-val pedig új érték írható bele. Megszakítást akkor váltanak ki, ha elérték a 0-t. Ezután az 1. számláló a beírt értékről, a 2. és a 3. pedig $FFFF-ről indul újra.

A számlálók írásakor arra kell ügyelni, hogy először az alsó, utána a felső byte-ot írjuk be.

Pontos időzítésre elsősorban szalagra íráskor és visszaolvasáskor van szükség. Itt az egyes jeleket impulzushosszuk azonosítja.

0 bit:	0.25 ms + 0.50 ms
1 bit:	0.50 ms + 0.25 ms
byte kezdet:	1.00 ms + 0.50 ms

A számlálók kiválóan használhatók véletlen szám generátornak is.

2.4.5. A RESET MŰKÖDÉSE

A gép jobb oldalán levő RESET gomb segítségével hozható a gép alapállapotba. Működési mechanizmusa némileg a megszakításokéhoz hasonló, azonban letiltani nem lehet.

A RESET benyomására a processzor felfüggeszti a működését mindaddig, amíg benyomva tartjuk. Kiengedéskor az utasításszámlálóba töltődik a $FFFC–$FFFD tartalma, tehát a $FFF6 és innen kezdődik a RESET program végrehajtása.

1) ROM bekapcsolás
2) SP = $FF; stack inicializálás
3) CLD ; bináris mód
4) ROM-ok indítása (JSR $CFA6)
   CBM azonosítójú ROM-okon JSR $8000
5) I/O-inicializálás (JSR $F30B)
6) STOP-lekérdezés

STOP nincs lenyomva:

7) RAM tesztelés, inicializálás (JSR $F352)
8) KERNAL vektorok írása
9) Video-inicializálás
10) BASIC indítás: JMP $8000

STOP lenyomva:

7) RAM-tesztelés, -inicializálás: csak bekapcsolásnál
8) KERNAL vektorok írása
9) Video-inicializálás
10) MONITOR (JMP $F445)

A RESET-tel azonos hatású a bekapcsolás is.

A RESET működésének módosítása hasonló módon lehetséges, mint a megszakításoké. Letiltani ugyan nem lehet, de a fent leírt RESET vektor átírásával az eredeti rutin kicserélhető. Ehhez gondoskodni kell arról, hogy állandóan a RAM legyen felülre lapozva (2.7. alfejezet). Ennek további következménye, hogy vagy teljesen önálló (RAM) megszakító rutint kell írnunk, vagy le kell tiltani a megszakításokat. Az átírásnak elsősorban programvédelemnél van jelentősége.

2.5. Billentyűzet

A billentyűzet használatát a gépkönyvek elég részletesen leírják. A billentyűk programból történő lekérdezéséről azonban csak annyit írnak, amennyi a GET és GETKEY utasításokhoz feltétlenül szükséges. Komolyabb program írásához azonban szükség van a billentyűzet-pufferelés ismeretére, más esetekben, pl. játékprogramoknál elkerülhetetlen egy-egy billentyű pillanatnyi lenyomottságának vizsgálata.

2.5.1. BILLENTYŰZETPUFFER

A billentyűzet folyamatos lekérdezését a hardvermegszakító rutin végzi. Amíg ezt a rutint le nem tiltjuk, az éppen lenyomott billentyű ASCII kódja egy átmeneti tárolóba, billentyűzetpufferbe kerül. A BASIC-interpreter innen veszi ki, mégpedig a beírás sorrendjében a karaktereket. A GET utasítás is innen olvas be.

A puffer a \$0527–\$0530 tárterületet foglalja el. A \$0527-es címen található a következő, még fel nem dolgozott billentyű kódja, és az utána következők az egyre magasabb címen találhatók. Ha a puffer megtelt, a lenyomott billentyűnek nincs hatása. A \$EF címen található egy mutató, ami a pufferbe került, de még fel nem dolgozott karakterek számát mutatja. Felhasználásával nagyon könnyű a puffert kiüríteni, csupán 0-t kell írni a \$EF címre. Erre gyakran van szükség, pl. ha a képernyőn lévő kérdésre a billentyűről várja a program a választ. A válasz bevitele előtt szokás kiüríteni a puffert.

Alaphelyzetben a hosszabb ideig nyomva tartott billentyűt ismételt lenyomásként értékeli a gép. Ez bizonyos esetekben hasznos lehet, de előfordulhat, hogy kifejezetten káros. A gép ismétlési tulajdonságai a \$0540-es cim tartalmának módosításával megváltoznak.

\$80 minden billentyű ismétlődik (alaphelyzet)
\$40 nincs ismétlés
\$00 csak a kurzorvezérlő billentyűk ismétlődnek

Az előzőekből kiderült, hogy a billentyűzetpuffer 10 byte hosszú. Ennél hosszabb nem lehet, de rövidebb igen, ha a \$053F címre 1 és 9 közötti számot írunk. Egyedül az 1-esnek van értelme olyan esetben, amikor a program egy-egy billentyű lenyomására vár bizonyos pontokon.

Mivel a billentyűkódokat a megszakítás automatikusan a pufferbe juttatja, a kérdés csak az lehet, hogy hogyan olvassuk ki onnan gépi programmal. Az eljárás az eddig leírtakból kikövetkeztethető:

- kivesszük a következő karaktert a \$0527-es címről
- a \$0528–\$0530-as tartományt 1-gyel lejjeb toljuk
- csökkentjük \$EF tartalmát 1-gyel

Mindezt a KERNAL GETIN (\$FFE4) nevű rutinja elvégzi. A billentyűzetpufferből az akkumulátorba olvassa a soronkövetkező karaktert. Működése a BASIC GET utasításhoz hasonló. 0-val tér vissza, ha a puffer üres. Használatakor

ügyeljünk arra, hogy ha az aktuális input egység nem a billentyűzet, akkor erről az egységről olvas be egy byte-ot.

Lehetőség van a billenyűzet lekérdezésére akkor is, ha a megszakítást letiltottuk. Az SCNKEY ($FF9F) rutin a lenyomott billentyű ASCII kódját a pufferbe tölti.

Mivel a billentyűzetpufferbe bármit írhatunk, tulajdonképpen szimulálhatjuk a valódi begépelést is. Pl. írhatunk olyan programot (akár BASIC, akár gépi programot), amelyik egy BASIC parancsot ír be a pufferbe. Amikor lefutott, végrehajtja a parancsot. Ezzel a módszerrel is elérhető, hogy egy program BASIC sorokat írjon be a tárba.

2.5.2. FUNKCIÓBILLENTYŰK

A funkcióbillentyűk használata BASIC-ben nem jelent problémát. A lenyomott funkcióbillentyű hatása ott teljesen ugyanaz, mintha begépeltük volna a megfelelő szöveget. Meglepőnek tűnhet ezután, hogy a funkcióbillentyűhöz kapcsolt szöveg egyáltalán nem kerül a billentyűzetpufferbe. Ennek az az oka, hogy ez a szöveg hosszabb is lehet 10 karakternél, és nem férne el a pufferben.

Az egyes billentyűkhöz rendelt szövegek leírására a $0567–$05E6 tartomány van fenntartva. A szavak sorban (F1-től HELP-ig), összefüggően, elválasztó jel nélkül vannak felsorolva. Hogy mégis el lehessen különíteni őket, a $055F–$0566 területen (8 byte), sorrendjüknek megfelelően le van írva az egyes szavak hossza.

Egy funkcióbillentyű lenyomásakor a megszakító rutin (az SCNKEY is) a $055D címre beírja a lenyomott billentyűhöz rendelt szó hosszát, a $055E-be pedig a szó első betűjének a $0567-es címtől számított relatív távolságát. Pl. a C 16-osnál a "GRAPHIC" (F1) szöveg kiírásánál a $055D-be 07 kerül, a $055E-be pedig 0. Gépi programból ezt is olvashatjuk a GETIN rutinnal byte-onként. Minden karakter kiolvasásakor a $055D-n levő érték csökken, a $055E értéke pedig nő 1-gyel, mindaddig, amíg a $055D értéke 0 nem lesz.

2.5.3. FÜGGETLEN LEKÉRDEZÉS: BILLENTYŰZETMÁTRIX

Elsősorban a játékprogramoknál szükséges az, hogy megtudja a program állapítani, hogy egy adott billentyű éppen lenyomott állapotban van-e, függetlenül az összes többi billentyűtől. Ezt az eljárást az alábbiakban ismertetjük.

Mindenekelőtt azt kell tudni, hogy a 64 db billentyű egy 8 x 8-as táblázatba, "mátrixba" rendezett, a következő ábra szerint. A lekérdezéshez először azt kell megnézni, hogy az illető billentyű a mátrix melyik sorában van. A K például a 4-es jelű sorban. Ebben az esetben a $FD30 és $FF08-as címekre egy olyan számot kell írnunk, amelynek csak a 4. bitje 0, a többi 1-es értékű. Ez az 11101111 ($EF) érték. Ezután olvasnunk kell az $FF08-as regisztert. A kiolvasott értékben az a bit lesz 0 értékű, amelyiknek megfelelő oszlopban lévő billentyű le van nyomva. Jelen esetben tehát a K lenyomottságát az jelzi, ha az 5. bit értéke 0. Ha csak a K van

lenyomva, akkor a beírások után az $FF08-ból $DF-et fogjuk kiolvasni. Ha $0F-et olvasunk, akkor egyszerre vannak lenyomva az M, K, O, N billentyűk.

Természetesen bármelyik másik sor is lekérdezhető, sőt egyszerre több is, ha a $FD30-as és a $FF08-as címre olyan számot írunk, amelyikben több bit értéke 0. Ekkor azonban a visszakérdezés már nem egyértelmű, bár gyakran ez is lehet a célunk.

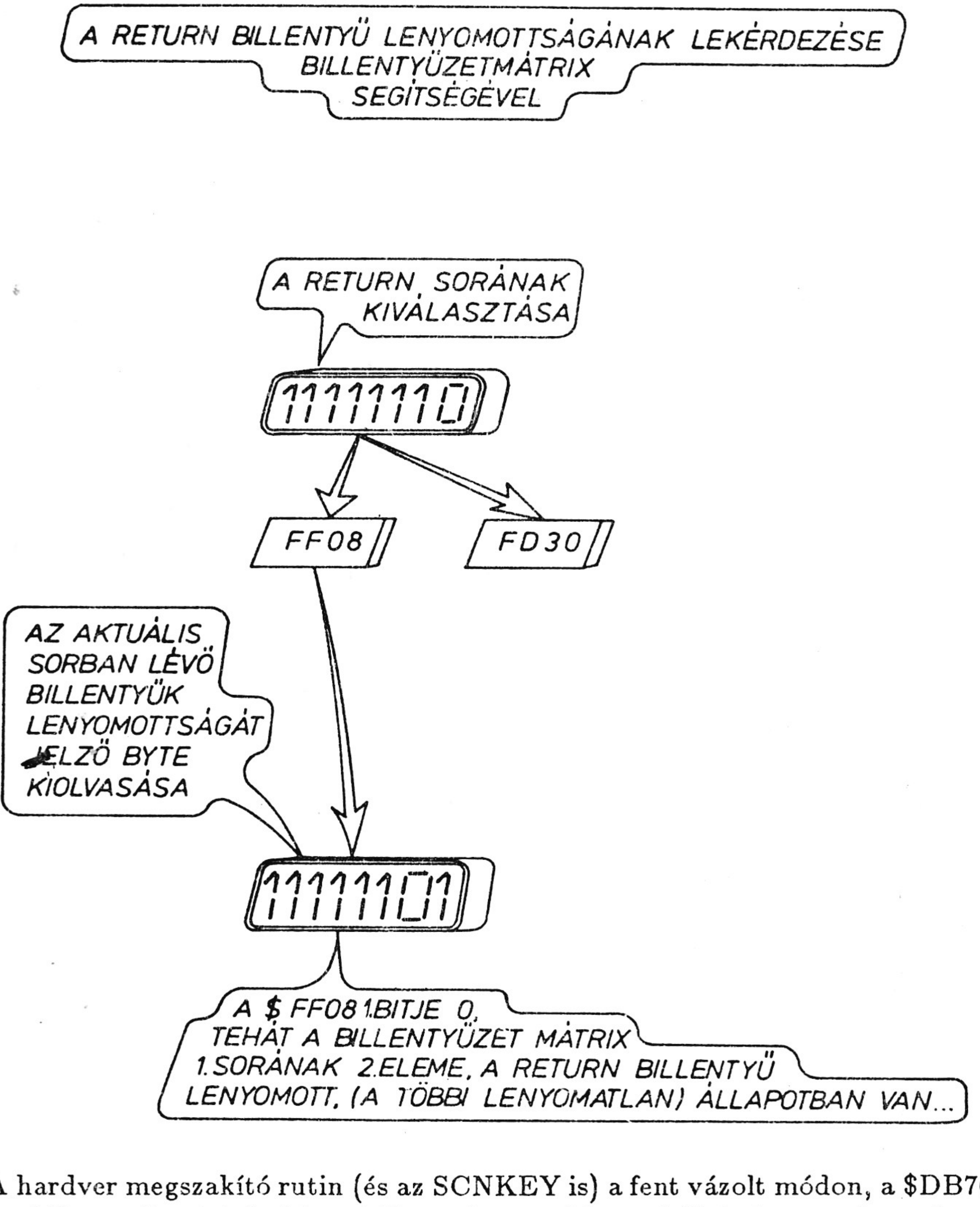
Az előbbiekben leírt eljárást segíti a $DB70-től kezdődő ROM-rutin.

```
DB70  0D  30  FD  STA  $FD30
DB73  3D  08  FF  STA  $FF08
DB76  AD  08  FF  LDA  $FF08
DB79  60          RTS
```

		kiolvasott							
		FE	FD	FB	F7	EF	DF	BF	7F
sor kiválasztás	bit	0.	1.	2.	3.	4.	5.	6.	7.
FE	0.	DEL	RETURN	£	HELP	F1	F2	F3	@
FD	1.	3 JOY 2 ↑	W JOY 2 ↓	A JOY 2 ←	4 JOY 2 →	Z	S	E	SHIFT JOY2TŰZ
FB	2.	5 JOY1	R JOY 1 ↓	D JOY 1 ←	6 JOY 1 →	C	F	T JOY1TŰZ	X
F7	3.	7	Y	G	8	B	H	U	V
EF	4.	9	I	J	0	M	K	O	N
DF	5.	CRSR ↓	P	L	CRSR ↑	> .	:	-	< ,
BF	6.	CRSR ←	*	;	CRSR →	ESC	=	+	/
7F	7.	1	HOME	CTRL	2	SPACE	C=	Q	STOP

A joystickek (botkormányok) állapotának lekérdezése szorosan kapcsolódik ehhez a témakörhöz, mert teljesen ugyanúgy történik, mint a billentyűké. (A billentyűzet-mátrixban a joystick pozíciókat is feltüntettük.) A különbség csak annyi, hogy ha a $DB70-es rutinba a $DB73-as címen lépünk be, akkor a billentyűk lekérdezése elmarad. Egyébként a lekérdezés párhuzamos, vagyis például a joy1 tűz gombja a T lenyomásával egyenértékű.

Bizonyára sokakat érdekel az, hogy hogyan lehet egy programot úgy átírni, hogy joystick helyett billentyűről működjön. Ezeknél az STA $FD30 marad ki, ezt kell pótolni. Vagy a JSR $DB73-at kell JSR $DB70-re átírni, vagy az LDA $FF08 helyére JSR $DB70-et írni. Írhatunk önálló lekérdező rutint is.



A hardver megszakító rutin (és az SCNKEY is) a fent vázolt módon, a $DB70-es rutint felhasználva kérdezi le a billentyűzetet. Most térjünk át arra, hogy hogyan rendel az egyes billentyűkhöz ASCII kódokat.

A ROM-ban az $E026–$E129 tartományban 4 db 8 x 8-as dekódoló táblázat van. Ezek pontos megfelelői a billenyűzetmátrixnak és az egyes billentyűkhöz rendelendő ASCII kódokat tartalmazzák az alábbiak szerint:

normál billentyűk	$E026–$E065
billentyűk SHIFT-tel	$E067–$E0A6
billentyűk C= -ral	$E0A8–$E0E7
billentyűk CTRL-lal	$E0E9–$E128

Tehát ezektől a táblázatoktól függ a billentyűzetkiosztás, amit sok esetben jó lenne megváltoztatni. Jó példa erre a Z és Y felcserélése, az írógépekhez szokottaknak ez sok problémát okoz. Egyszerű fogással ez sajnos nem érhető el, csak a dekódoló rutin egy részének újraírásával. A dekódoló rutin kezdetére a $0545-$0546-os címen lévő mutató mutat. Alapértéke $DB7A.

2.6. Hanggenerálás

A C 16-os és a Plus/4-es hanggenerálási lehetőségei sajnálatos módon sokkal szegényebbek, mint a C 64-esé. Így be kell érni a szerényebb kialakítással, de azért itt is van lehetőség néhány érdekes megoldásra.

A C 16-osban és a Plus/4-esben a hanggenerálás is (mint oly sok minden) a TED chip feladata. Itt két, egymástól függetlenül programozható hanggenerátor áll rendelkezésünkre. BASIC-ből nagyon egyszerű egy hangot megszólaltatni, gépi kódból azonban több regisztert is be kell állítani ahhoz, hogy egy hang úgy szólaljon meg, ahogy mi szeretnénk. A két hanggenerátor közül az első csak négyszögjelet tud szolgáltatni, itt tehát hullámforma választási lehetőség nincs. A második választhatóan négyszögjel vagy fehérzaj előállítására alkalmas.

Egy hang megszólaltatásához három regisztert kell beállítani. Ebből kettő a frekvenciát határozza meg. A harmadikban lehet a hangerőt megadni, ill. a hangot ki- vagy bekapcsolni.

Ha az első hanggenerátort szeretnénk megszólaltatni, akkor először állítsuk be a frekvenciát. A TED $FF0E címen lévő regisztere tartalmazza az első hanggenerátor frekvenciaértékének alsó byte-ját, míg a felső byte a $FF12 címen helyezkedik el. Ügyeljünk azonban arra, hogy a $FF12 címen grafikai információk is vannak, a hang csak a 0–1. biteken foglal helyet. Tehát a frekvencia felső byte állításakor csak ezt a két bitet szabad megváltoztatni. Az alsó ill. felső byte különböző frekvenciákhoz tartozó értékeit a 6. táblázat tartalmazza.

Ha beállítottuk a frekvenciát, akkor utána a $FF11 címen lévő hangvezérlő regiszterben kell a hangerőt beállítani, majd a hangot bekapcsolni. Ez a hangvezérlő regiszter mindkét csatornához tartozik, így a hangerő csatornánként nem különbözhet. Ennek a regiszternek az alsó négy bitje tartalmazza a hangerő értékét, ami azonban csak 8-ig hatásos. Ez az érték a leghangosabb, ennél nagyobb szám beírása nem eredményez további hangerőnövekedést. A negyedik bit (16-os helyiértékű) ha 1, akkor az első hanggenerátor be van kapcsolva, ha 0, akkor kikapcsolt. Tehát nekünk az alsó négy bitben kell a hangerőt beállítani, majd a negyedik bit magasra állításával a hangot megszólaltatni. A következő kis példaprogram a normál A hangot (440 Hz) szólaltatja meg 7-es hangerővel az 1. hanggenerátoron:

```
HANG  LDA  #$02   ;frekv.alsórész ért.
      STA  $FF0E  ;1. hanggen. frekv. alsó byte reg.
      LDA  $FF12  ;1. hanggen. frekv. felső byte reg.
```

```
AND  #$FC   ;0-1. bitek nullázása
ORA  #$03   ;0-1. bitek írása
STA  $FF12  ;új érték vissza
LDA  #$17   ;1. gen. bekapcsolás 7-es hangerővel
STA  $FF11  ;hangvezérlő reg.-be
BRK
```

A programunk a RESET gomb megnyomásáig szolgáltatja a normál A hangot. A hang kikapcsolása programból a $FF11 negyedik bitjének alacsonyra állításával lehetséges. Ha kiegészítjük az előző kis programot az itt következő résszel, akkor a hang a STOP billentyű lenyomására elhallgat. A kiegészítés első utasítását a BRK helyére kell beírni.

```
STOP  BIT  $91    ;STOP gomb figyelése
      BMI  STOP   ;nincs lenyomva
      LDA  $FF11  ;hang vezérlő reg.
      AND  #$EF   ;4.bitjének kikapcsolása
      STA  $FF11  ;új érték vissza
      BRK
```

A második hanggenerátor megszólaltatása ugyanilyen eljárással lehetséges, csak a beállítandó regiszterek címe más. A második hanggenerátor frekvencia alsó rész a $FF0F címen van, míg a felsőrész a $FF10 címen lévő regiszter 0.–1. bitjei. A hangerőt és a bekapcsolást itt is a $FF11 címen lehet beállítani, itt azonban megkülönböztetjük a négyszögjel és a fehérzaj bekapcsolását. A hangerő az első hanggenerátornál már említett alsó négy bit. A négyszögjel engedélyezése az 5. bit magasra állításával, a fehérzaj bekapcsolása pedig a 6. bit magasra állításával lehetséges. Ha mindkettőt magasra állítjuk, a négyszögjel fog megszólalni. Lehetőség van mindkét regiszter együttes megszólaltatására is. Ilyenkor a regiszterekhez tartozó frekvenciaértékek beállítása után a $FF11 4. és 5. bitjének magasra állításával mindkét regiszter négyszögjelet szolgáltat, de ha a 4. és a 6. bitet állítjuk magasra, akkor az első hanggenerátor négyszögjelet, a második fehérzajt szólaltat meg.

A $FF11 hangvezérlő regiszter 8 bitjéből hetet már ismerünk. Nézzük most meg a legfelső, 7. bit szerepét. Ennek értéke alapállapotban 0. Ilyenkor egy négyszögjel bekapcsolásakor a kimeneten megjelenő hang hullámhossza a frekvenciától, amplitudója a hangerőtől függ. Ha a $FF11 regiszter 7. bitjét 1-re állítjuk, akkor kikapcsoljuk a beépített négyszögjel-generátort, és a hang kimeneten a hangerővel arányos egyenszint jelenik meg. Ezt az egyenszintű jelet azonban nem lehet hallani. Úgy lehet hallhatóvá tenni, hogy a hangerő megfelelő ütemű változtatásával mi formáljuk a hullámalakot. Így tetszőleges hullámformák és igen érdekes hanghatások jöhetnek létre. A következő kis program ezt a lehetőséget használja ki:

```
. 200D  A9  97      LDA  #$97    ;1.hanggen. analóg jel
. 200F  8D  11  FF  STA  $FF11   ;7-es hangerővel bekapcsolva
. 2012  20  30  20  JSR  $2030   ;időhúzás
. 2015  AD  11  FF  LDA  $FF11   ;hangerő
. 2018  29  F0      AND  #$F0    ;nullázása
. 201A  8D  11  FF  STA  $FF11
. 201D  20  30  20  JSR  $2030   ;időhúzás
. 2020  AD  11  FF  LDA  $FF11   ;hangerő
. 2023  09  08      ORA  #$08    ;maximumon
. 2025  8D  11  FF  STA  $FF11
. 2028  4C  12  20  JMP  $2012   ;ugyanez elölről

. 2030  A0  FE      LDY  #$FE    ;időhúzó kettős ciklus
. 2032  A2  00      LDX  #$00
. 2034  E8          INX
. 2035  D0  FD      BNE  $2034
. 2037  C8          INY
. 2038  D0  F8      BNE  $2032
. 203A  60          RTS
```

Ha egy hangot csak meghatározott ideig szeretnénk megszólaltatni, akkor egy bizonyos idő eltelte után ki kell kapcsolni a hangvezérlő regiszter megfelelő bitjét. Programból való időzítése igen pontatlan és nem is mindig alkalmazható, ha közben még mást is kell csináltatni a programmal (pl. a képernyőt kezeltetni). Erre az időzítésre nagyon jól használhatók a rendszerváltozók területén lévő számlálók. Ezeket a megszakító rutin figyeli, és ennek alapján elvégzi a megfelelő hanggenerátor kikapcsolását. Tehát a bekapcsolásról nekünk kell gondoskodni, a megszakító rutin csak kikapcsol.

Az időzítő számlálók a következők:

1. hanggenerator	alsó	byte –	$04FC
	felső	byte –	$04FE
2. hanggenerator	alsó	byte –	$04FD
	felső	byte –	$04FF

A számlálók növelése 1/50 s-onként történik. Ha valamelyik értéke eléri a 0-t, akkor a megszakító rutin a számlálóhoz tartozó hanggenerátort kikapcsolja. Mivel a számlálók értéke növekszik, ezért az időtartam kettes komplemensét kell a számlálóba beírni. Így minél nagyobb számot írunk be, annál hamarabb fogja a $FF után a 0-t elérni. A következő példaprogram ezt az időzítési megoldást tartalmazza:

```
. 2000  A9  02      LDA  #$02
. 2002  8D  0E  FF  STA  $FF0E
. 2005  AD  12  FF  LDA  $FF12
. 2008  09  03      ORA  #$03
. 200A  8D  12  FF  STA  $FF12
. 200D  A9  17      LDA  #$17
. 200F  8D  11  FF  STA  $FF11
. 2012  A9  80      LDA  #$80   ;kb. 2.5 másodperc
. 2014  8D  FC  04  STA  $04FC  ;beállítása
. 2017  A9  FF      LDA  #$FF   ;számláló felső byte
. 2019  8D  FE  04  STA  $04FE  ;beállítása
. 201C  00          BRK
```

Lefuttatáskor megfigyelhetjük, hogy a program már rég megállt, a hang még mindig szól. Csak akkor hallgat el, ha a számláló lefutott 0-ra. Ez alatt az idő alatt bármilyen hasznos programot futtathatunk.

2.7. Tárfelosztás, memórialapozás

A C 16-os és a Plus/4-es tárja elsősorban mennyiségében különbözik egymástól. Míg a C 16-osban 16K RAM van, addig a Plus/4-es közel 64K RAM-mal rendelkezik. Ez elég jelentős különbség és főleg grafikát használó programoknál szembeötlő. Ma már azonban könnyen hozzájuthatunk olyan tárbővítőhöz, mellyel a C 16-os tárját is 64K-ra bővíthetjük.

Először nézzünk egy vázlatos tártérképet, ami mindkét gépre érvényes:

RAM	$0000–$07FF	:rendszerváltozók
	$0800–$0BFF	:színmemória
	$0C00–$0FFF	:képernyőmemória
bittérkép használatakor:		
	$1800–$1BFF	:fényerő
	$1C00–$1FFF	:szín
	$2000–$3FFF	:bittérkép
ROM	$8000–$CFFF	:BASIC ROM
	$D000–$D3FF	:nagybetűs karakterkészlet
	$D400–$D7FF	:kisbetűs karakterkészlet
	$D800–$FFFF	:KERNAL
	$FD00–$FEFF	:I/O
	$FF00–$FF40	:TED
	$FF41–$FFFF	:ugrótáblák

Az alapkiépítésű C 16-osban a RAM a $0000–$3FFF területre terjed ki. A BASIC programok számára tehát a $1000–$3FFF tartomány szabad.

A Plus/4-esnél, és a tárbővítős C 16-osnál a RAM a $0000–$FFFF területre terjed ki, melyből az $FD00–$FF40-ig terjedő rész semmiképpen sem használható. A BASIC program a $1000–$FCFE területet használja.

Látható, hogy a ROM és a RAM a $8000–$FFFF területen "párhuzamos", tehát az adott címeken ROM is és RAM is van. Ahhoz, hogy mindkettőt használni tudjuk, szükséges a memórialapozási technika elsajátítása.

2.7.1. RAM–ROM LAPOZÁS

Felmerülhet a kérdés, hogy a $8000–$FFFF tartományban, ahol ROM is és RAM is van, pl. egy LDA utasítás melyikből olvas. Alapállapotban nem tudunk a monitorral $8000 fölött programot írni. Itt a D parancs is a BASIC ROM-ot disassemblálja, és a $2000 címen futó LDA $8000 is a ROM-ból olvas. Az író utasítások ellenben mindig RAM-ra vonatkoznak.

A RAM "felülre" lapozása az

STA $FF3F

utasítással lehetséges. (Teljesen mindegy, hogy mit írunk a $FF3F-be, mert az átváltást maga a beírás váltja ki.) Hatására – a visszaváltásig – minden utasítás a RAM-ra fog vonatkozni. Használatakor arra kell a legjobban ügyelni, hogy a megszakítás le legyen tiltva, ellenkező esetben a megszakító rutin is a RAM-ban próbál futni. Kivétel természetesen az, ha van saját megszakító rutinunk a RAM-ban.

A ROM felülre lapozása az

STA $FF3E

utasítással lehetséges. Ekkor ismét visszaáll a bekapcsolás utáni állapot; az olvasó utasítások a ROM-ra, az író utasítások pedig a RAM-ra vonatkoznak.

A memórialapozás csak a tár "felső felét" ($8000–$FFFF) érinti. A $0000–$7FFF területeken elkerülhetetlenül RAM van. A lapozó utasítás csak ezen a területen futhat, ellenkező esetben saját magát, a folytatását lapozná el.

A rendszerváltozók területén több olyan rövid szubrutin van, amelyik a "ROM alól", tehát RAM-ból olvas az akkumulátorba egy byte-ot. Végrehajtás után ismét a ROM-ot kapcsolják felülre.

```
04A5  78           SEI
04A6  8D  3F  FF   STA  $FF3F
04A9  B1  3B       LDA  ($3B),Y
04AB  8D  3E  FF   STA  $FF3E
04AE  58           CLI
04AF  60           RTS
```

Ez a $3B–$3C indirekt címet használva olvas a RAM-ból.

A következő rutin általánosan használható a ROM alól olvasáshoz.

```
0494  8D  9C  04  STA  $049C
0497  78          SEI
0498  8D  3F  FF  STA  $FF3F
049B  B1  00      LDA  ($00),Y
049D  8D  3E  FF  STA  $FF3E
04A0  58          CLI
04A1  60          RTS
```

Meghívás előtt azt az indirekt címet kell az akkumulátorba tölteni, amelyik az elhozandó értékre mutat. A rutin az akkumulátorba tölti a kívánt RAM-tárcím tartalmát.

Ez a program azért is érdekes, mert önmagát írja felül. Az LDA ($00),Y utasításban a zárójelben levő indirekt cím tulajdonképpen nem 0, hanem az, amit az első sorban lévő STA $049C beleír. Tehát ha az AC = $2B-vel hívjuk meg, akkor LDA ($2B),Y hajtódik végre.

Alapkiépítésű C 16-osnál az első pillanatban úgy tűnhet, hogy semmi értelme sincs a RAM–ROM kapcsolásnak. (Ez egyébként BASIC program futásánál nagyon is igaz.) Ugyanis ha a ROM-ot lelapozzuk, kérdéses, hogy mi marad a helyén, hiszen a $4000–$FFFF tárterületen RAM nincs. Azonban ezen a területen is RAM-ot érzékelünk, méghozzá ugyanazt, amelyik a $0000–$3FFF területen van. A címtartományok, és a nekik megfelelő RAM-területek a következők:

$4000–$7FFF = $0000–$3FFF
$8000–$BFFF = $0000–$3FFF
$C000–$FFFF = $0000–$3FFF

Ez azt jelenti, hogy az STA $5125 teljesen egyenértékű az STA $1125-tel. Ennek a ténynek köszönhető, hogy az alapkiépítésű C 16-osnál is van RAM terület párhuzamosan a legkényesebb $FF80–$FFFF ROM területtel.

A fenti felosztás alól kivétel a $0000 és a $0001-es cím, mivel ezek a processzor kapuregiszterei. A "valódi" $0000–$0001 címek csak a $4000–$4001 címen keresztül érhetők el.

2.7.2. ROM–ROM LAPOZÁS

Plus/4-esnél 2 db 32 K-s ROM van párhuzamosan a $8000–$FFFF területen. Az egyikben a BASIC ROM és a KERNAL, a másikban 3 beépített felhasználói program van. A C 16-osban mindennek csupán a lehetősége van meg. Ezen kívül még 2 külső (cartridge) ROM használatára van lehetőség.

Ha a $FF3E-vel a ROM-ot választottuk ki, akkor a 4 lehetséges ROM-ot 16-

féle kombinációban használhatjuk. Ehhez a RAM-lapozáshoz hasonlóan a $FDD0–$FDDF tartomány valamelyik címét kell írnunk.

Ezek a ROM-ok a következők:

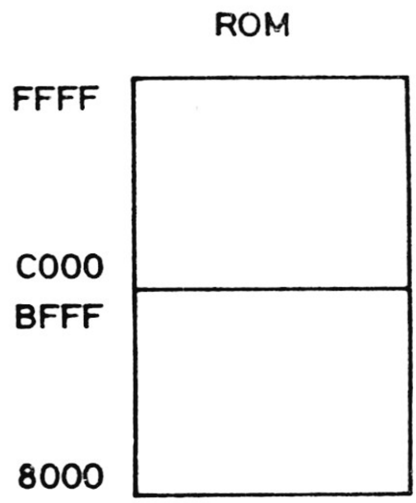
Sorszám	Név	Rövidítés
0	Rendszer ROM	S
1	Belső bővítő ROM	E
2	Cartridge 1	C1
3	Cartridge 2	C2

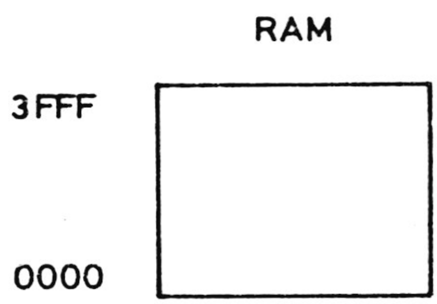
Mindegyikük alsó és felső 16 K-s darabra osztható. A rendszer ROM-nál ezeket $8000–$BFFF-ig BASIC ROM-nak, $C000–$FFFF-ig KERNAL-ROM-nak nevezik. Valójában a BASIC-nek egy része átlóg a KERNAL-ba.

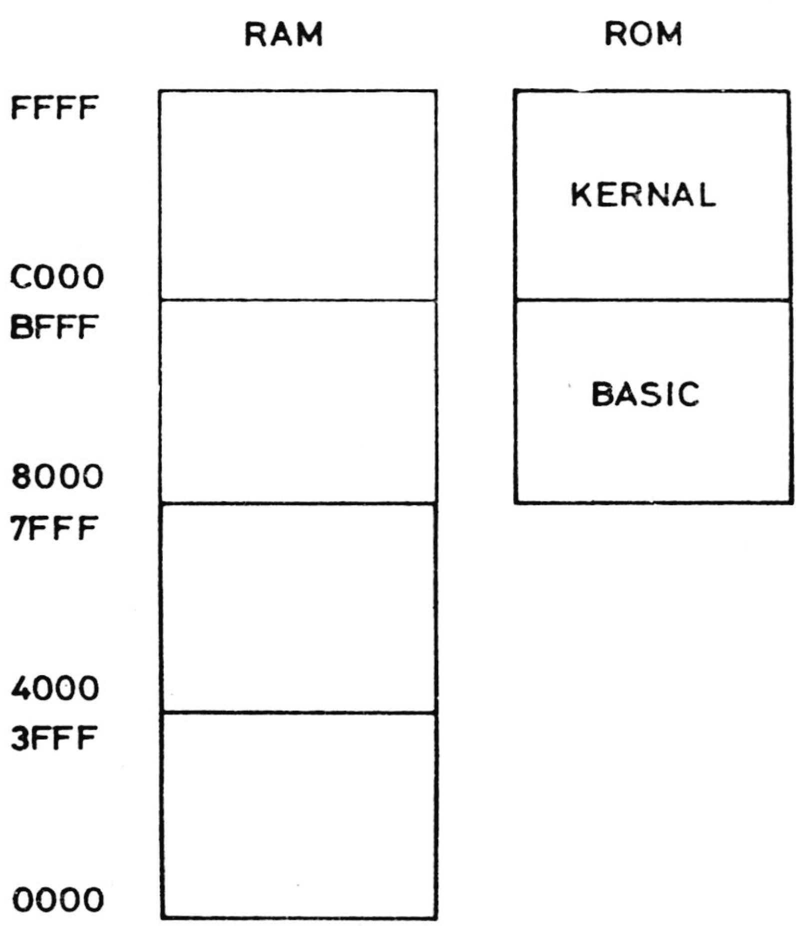
ROM lapozási lehetőségek:

Cím	ROM-BANK kód	$8000–$BFFF	$C000–$FFFF
$FDD0	0	S	S
$FDD1	1	E	S
$FDD2	2	C1	S
$FDD3	3	C2	S
$FDD4	4	S	E
$FDD5	5	E	E
$FDD6	6	C1	E
$FDD7	7	C2	E
$FDD8	8	S	C1
$FDD9	9	E	C1
$FDDA	10	C1	C1
$FDDB	11	C2	C1
$FDDC	12	S	C2
$FDDD	13	E	C2
$FDDE	14	C1	C2
$FDDF	15	C2	C2

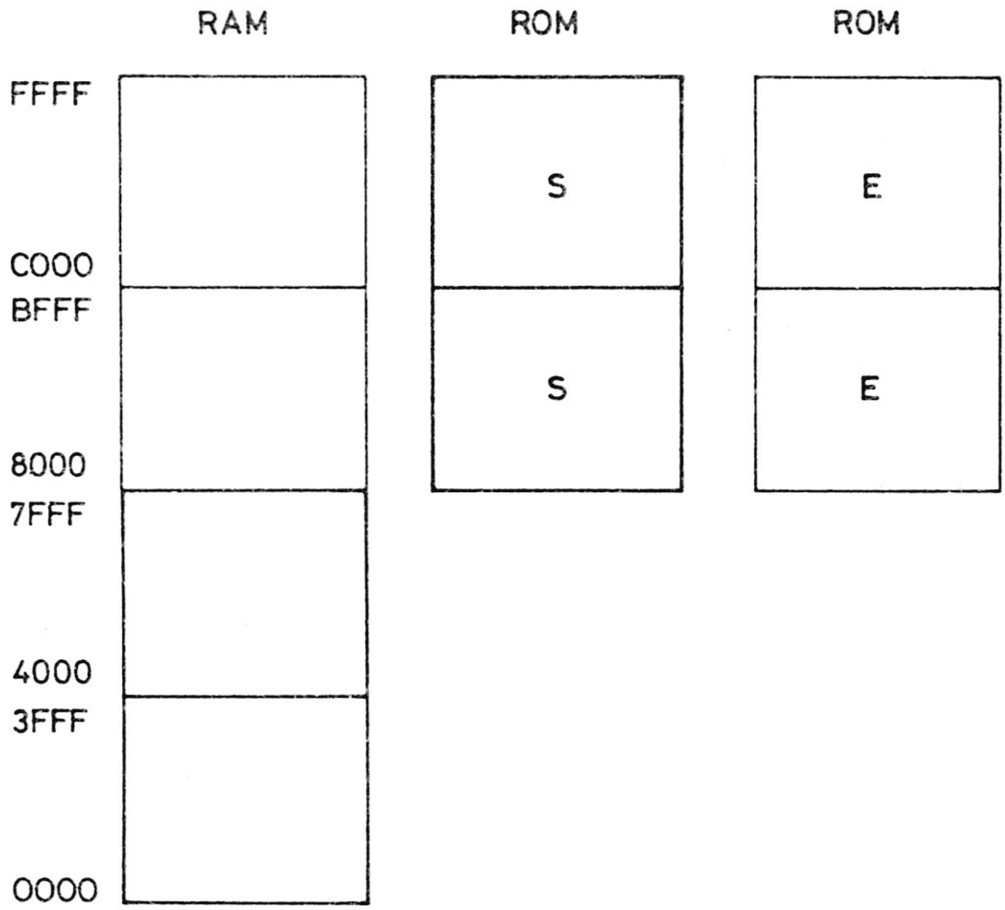
A következő ábrán az alapkiépítésű és a bővített C 16-os tártérképét mutatjuk be.


ROM
FFFF
C000
BFFF
8000

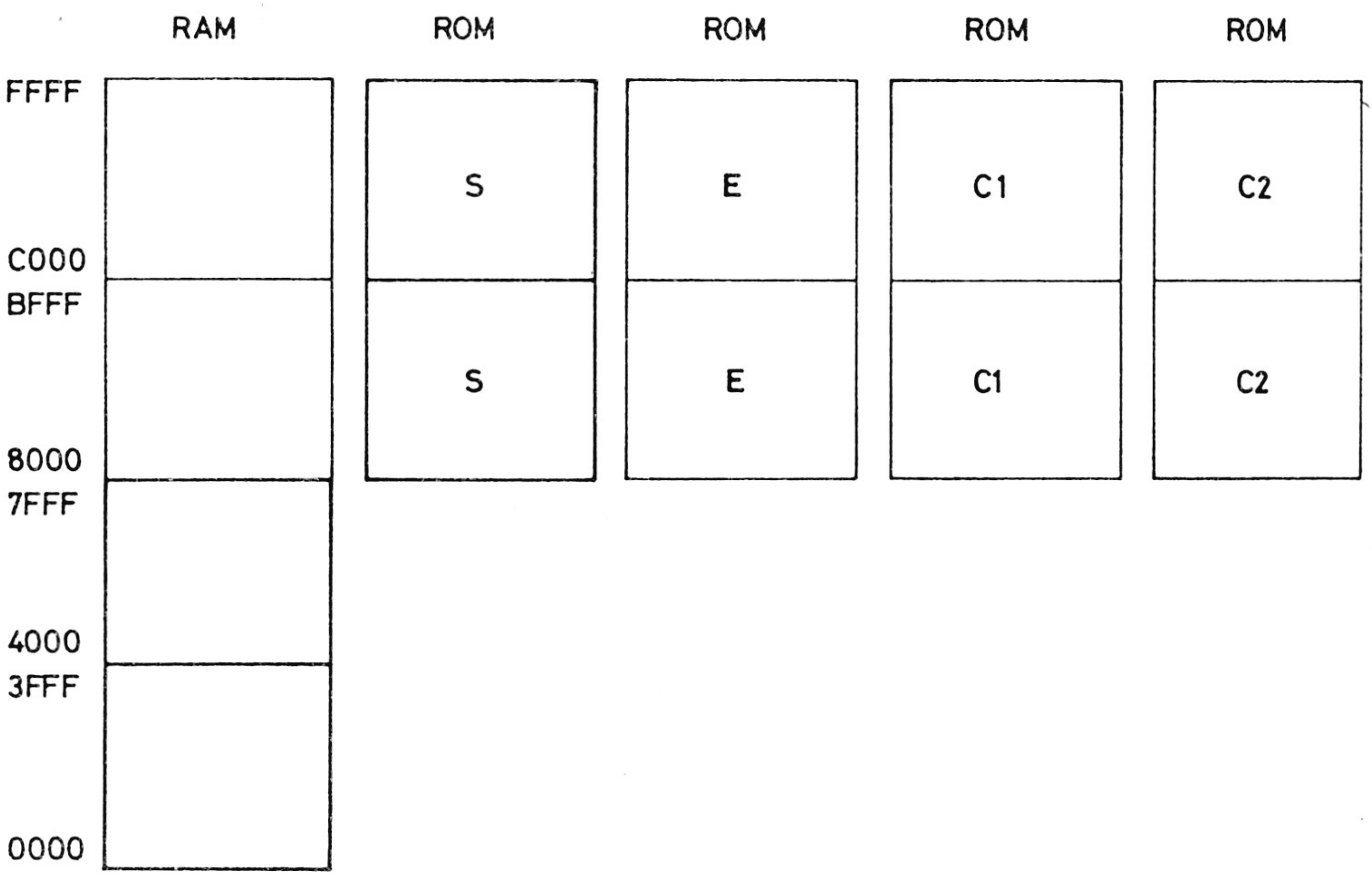

RAM
3FFF
0000


RAM
ROM
FFFF
KERNAL
C000
BFFF
BASIC
8000
7FFF
4000
3FFF
0000

A bemutatott ábrán az alapkiépítésű Plus/4-es tártérképét láthatjuk.



A C 16-os és a Plus/4-es teljes RAM–ROM használatának lehetőségei.



A monitor egyébként csak a RAM–ROM átkapcsolásra van felkészítve. Ha a $07F8 címen a 7. bit 0, akkor a ROM-ot, ha pedig 1, akkor a RAM-ot használja minden funkcióban. A ROM-ok közül természetesen mindig azt, amelyiket legutoljára kiválasztottuk.

Bármelyik ROM-nak a $8000–$BFFF-ig terjedő része tanulmányozható a beépített monitorral, mert a monitorprogram $C000 felett helyezkedik el. Ezt az teszi lehetővé, hogy a megszakító rutin olyan ROM-felosztást lapoz, amilyen ROM-BANK kódot a $FB számára kijelöl. Ha ide például 1-et írunk (csak Plus/4-esnél), akkor a KERNAL és a beépített bővítő ROM alsó ($8000–$BFFF) része van jelen. Így még a bővítő ROM rutinjainak próbafuttatására is lehetőség van.

Ha a felső területet akarjuk tanulmányozni, akkor az adott területet át kell másolni RAM-ba. Ehhez egy viszonylag egyszerű program megírása szükséges, amiben segítségünkre lehet a $FCF7 címen kezdődő "LONG FETCH" rutin, amelyik a ($BE),Y szerint olvas be egy byte-ot az akkumulátorba a másik ROM-ból. Meghívása előtt az X-regiszterbe kell tölteni azt a ROM-BANK kódot, amelyikből olvasni akarunk, az akkumulátorba pedig azt, amelyikbe vissza szeretnénk térni.

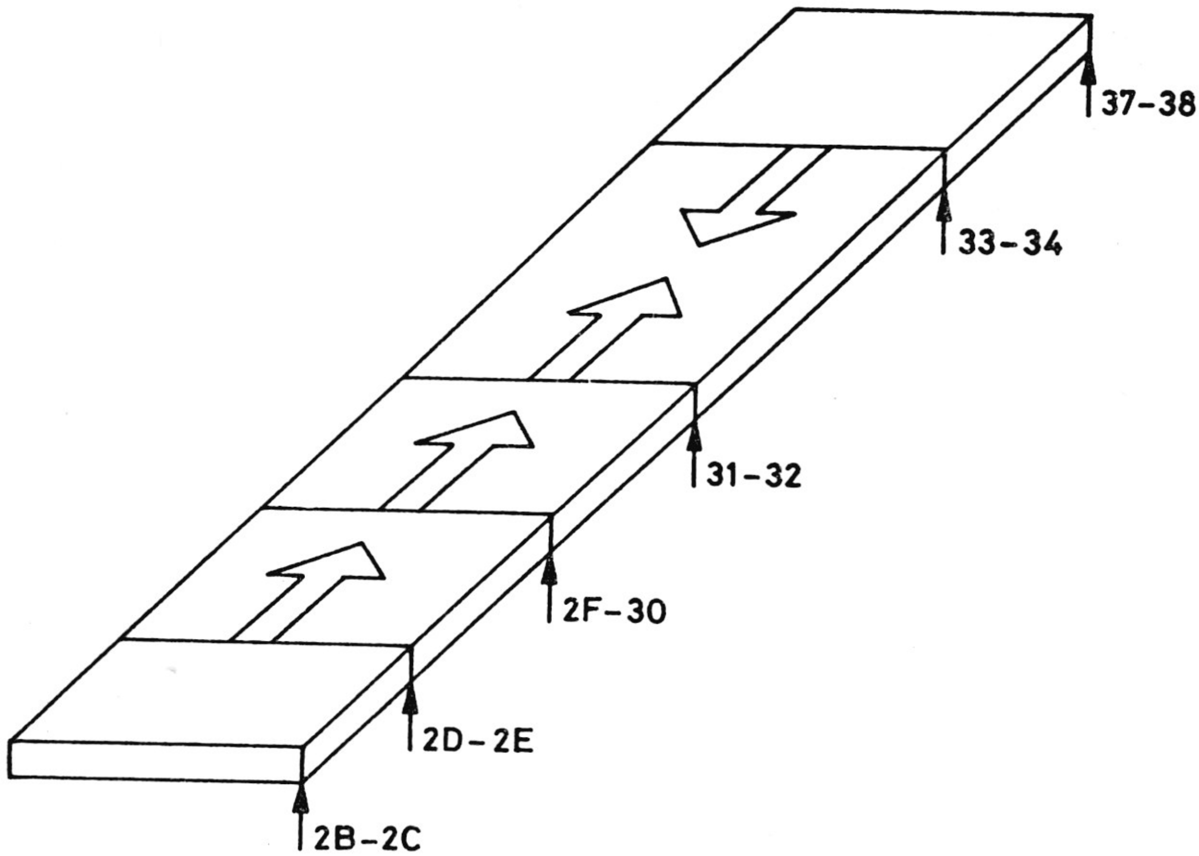
A tár $FC00–$FCFF tartománya az összes ROM számára közös terület, tehát nem lapozható. Emiatt itt találhatjuk a legfontosabb ROM-kezelő rutinokat. Az sem véletlen, hogy az IRQ is itt kezdődik és itt is fejeződik be.

2.8. BASIC programok és változók tárolása

Ebben a fejezetben gépi kódú szemmel vizsgáljuk meg, hogyan tárolja a C 16-os és a Plus/4-es számítógép a BASIC programokat és változókat a tárban. A BASIC programok és változók tárolására a BASIC RAM, azaz a programozáshoz rendelkezésre álló szabad tár alkalmas. Ez a BASIC munkaterület C 16-osnál $1001-től $4000-ig, míg a Plus/4-esnél $1001-től $FD00-ig terjed. Azért nem $1000-tól indul, mert ezen a címen mindig 0-nak kell lennie, más érték az interpretert megzavarhatja. Ha nem 0 a $1000 értéke, akkor pl. a RUN utasítás SYNTAX ERROR hibaüzenetet eredményez. Most nézzük meg, hogyan van felosztva a BASIC munkaterület egy BASIC program jelenléte közben. A pillanatnyi felosztást a rendszerváltozók területén lévő mutatók határozzák meg.

BASIC mutatók

$2B–$2C BASIC program kezdete mutató, de ez jelöli az egész BASIC munkaterület kezdetét is. Ettől a címtől kezdődően kerülnek a tárba az általunk beírt BASIC programsorok. Ha a mutató értékét feljebb állítjuk, akkor az egész BASIC munkaterület kezdete feljebb kerül. A GRAPHIC utasítás pl. Plus/4-esnél ezt a mutatót helyezi $4000-ra a bittérkép fölé, s emiatt az egész szabad tár lecsökken.



Írjuk be pl. a következő két BASIC programsort:

```
10 PRINT "HELP"
20 END
```

Ennek a két sornak az ábrázolása a tárban a következő:

```
>1000   00
>1001   [0E 10]   [0A 00]   [99] 20   22  48  45  4C  50  22  [00]
                    10                "   H   E   L   P   "

>100E   [14 10]   [14 00]   [80] [00]
                    20

>1014   00 00
```

A BASIC sor felépítése a következő: az első két byte egy mutató, ami a következő BASIC sor abszolút címét tartalmazza fordított sorrendben, tehát alsó–felső byte szerint. A következő két byte a BASIC sorszámot tartalmazza szintén alsó–felső byte sorrendben. Ezután következik a BASIC sor leírása. A BASIC utasításokat tokenjeik helyettesítik, a többi karakter az ASCII kódján szerepel. A sort mindig egy 00 byte zárja. A BASIC program végét egy olyan abszolút mutató jelzi, melynek mindkét byte-ja 00.

$2D–$2E Változóterület kezdete mutató. Ezen a területen kerülnek ábrázolásra az általunk használt változók. A C 16-os és a Plus/4-es négyféle változót tud

megkülönböztetni:
- valós típusút,
- egész típusút,
- füzér (sztring) típusút,
- függvénydefiníciót.

A változók nevének 2 byte van fenntartva, értéküknek pedig 5. Így egy változó ábrázolásához 7 byte-ra van szükség. Mivel egy változó nevét csak 2 byte-on lehet ábrázolni, ezért csak kétbyte-os változóneveket célszerű használni. Hosszabb változónév is megengedett, de ezeknek is csak az első két karaktere kerül a tárba. Ha ez a két karakter megegyezik, a többi már hiába különbözik, a két változó ugyanazt jelenti. A változó típusa a változónév ábrázolásában különbözik, méghozzá a név számára fenntartott két byte legfelső, hetedik bitjében. Itt még szerepet játszik az is, hogy egybetűs vagy kétbetűs változóneveket használtunk. Az ábrázolási kódok a következők:

		A változónév bitjei
Valós típ. változó	– egybetűs:	0XXXXXXX 00000000
	– kétbetűs:	0XXXXXXX 0XXXXXXX
Egész típ. változó	– egybetűs:	1XXXXXXX 10000000
	– kétbetűs:	1XXXXXXX 1XXXXXXX
Füzér típ. változó	– egybetűs:	0XXXXXXX 10000000
	– kétbetűs:	0XXXXXXX 1XXXXXXX
Függvénydefiníció	– egybetűs:	1XXXXXXX 00000000
	– kétbetűs:	1XXXXXXX 0XXXXXXX

Az X-ek helyén a változó név megfelelő betűjének az ASCII kódja van.

Valós típ. változó – jelölése: –
– hossza : 7 byte
pl. AB = 1

41 42	81 00 00 00 00
NÉV	ÉRTÉK

Az első két byte a név kódját tartalmazza a változó típusa szerint. A következő öt byte-on a változó értéke kerül ábrázolásra lebegőpontos alakban.

Egész típ. változó – jelölése: %
– hossza : 7 byte
pl. AB% = 1

C1 C2	00 01	00 00 00
NÉV	ÉRTÉK	ÜRES

Az első két byte itt is a nevet tartalmazza a változótípusnak megfelelő kódban, a következő két byte-on a változó értéke szerepel binárisan, fixpontosan.

Füzér típ. változó – jelölése: $
– hossza : 7 byte
pl. AB$ = "A"

41 C2	01	FA FC	00 00
NÉV	H.	KEZD.	ÜRES

A füzérek ezen a változóterületen közvetlenül nem szerepelnek, mert nekik egy külön füzérterület van fenntartva a BASIC munkaterület végén. Itt csak egy mutatót kapnak, ami erre a területre mutat. Az első két byte itt is a nevet tartalmazza. A következő byte a füzér hosszát mutatja meg a kezdőcímtől visszafelé haladva. Mivel a hossz egybyte-os, ebből következik, hogy egy füzér max. 255 karakter hosszú lehet. Ha ennél hosszabb, a STRING TOO LONG hibaüzenetet kapjuk (BASIC-ben). A következő két byte egy kétbyte-os mutató alsó–felső byte sorrendben. Ez mutatja meg a füzérterületen az aktuális füzér kezdőcímét.

Függvénydefiníció – jelölése: FN
– hossza : 14 byte
pl. FN AB(ZX)

C1 42	0E 10	27 10 5A	5A 58	81 00 00 00 00
NÉV	KÉPL.	MUT.ÉR.	VÁ.NÉV	VÁLT.ÉRTÉK

Az előzőekben leírt változótípusok mind 7 byte-on kerültek ábrázolásra. A függvénydefiníció ennek pont a kétszeresét veszi igénybe. Az első két byte itt is a nevet tartalmazza (AB). Ezt követi egy kétbyte-os mutató, ami arra a képletre mutat, ahol a függvényt definiáltuk a programban. Az ezt követő két byte a változóterület azon helyére mutat, ahol a függvény változójának (ZX) értéke van tárolva. Ezt egy kitöltő karakter követi, hogy kiegészítse 7 byte hosszúra az első részt. Ez a karakter a változónév első karaktere (Z). A következő két byte a független változó nevét tartalmazza (ZX), majd ezt követi a változó értéke a típustól függő alakban (jelen esetben lebegőpontosan).

$2F–$30 Tömbkezdet mutató. E mutatótól kezdve kerülnek a tárba a tömbváltozók. Ha a program futás közben találkozik egy tömbdefiniálással, pl.

DIM AB(1,2)

akkor ezen a területen állítja fel a táblázatot.

41 42	27 00	02	00 03	00 02
NÉV	REL.	DI.	DIM.	DIM.

A tömbváltozóra vonatkozó adatok és mutatók 5 byte-ot foglalnak le a tömbterület kezdetén, majd még annyiszor kettőt, ahány dimenziós a tömb. Jelen példánkban még 2*2-t, mert a tömbünk kétdimenziós. Ezután kerül ábrázolásra egyenként a tömbelemek értéke. A tömbváltozóra vonatkozó adatok első két byte-ja tartalmazza a tömbváltozó nevét (AB), természetesen a változótípusra vonatkozó kódolás szerint. A következő két byte egy relatív mutató. Megmutatja, hogy az aktuális tömbdefinícióhoz képest mennyivel feljebb kezdődik a következő tömb definiálása. Egyszerűbben fogalmazva megadja, hány byte hosszú az aktuális tömb definícióval és adatokkal együtt. Az ezt követő byte értéke megmutatja, hány dimenziós a tömb. Ez jelen példánkban 2, hiszen a DIM után kétféle dimenziót adtunk meg. Az ezután következő byte-párok az egyes dimenziók nagyságát adják meg, az utolsó dimenziótól visszafelé haladva. Ügyeljünk arra, hogy itt nem a szokásos kétbyte-os mutató szerint, tehát alsó–felső byte sorrendben van ábrázolva az érték, hanem először a felső byte, majd ezt követi az alsó byte. Jelen példánkban az utolsó dimenzió nagysága 3 (0–2), tehát a két byte értéke 00 03. Az első dimenzió nagysága 2 (0–1), tehát a következő két byte 00 02. Ezt követően kerülnek ábrázolásra a tömbelemek értékei, a tömb típusától függő alakban. Ez az alak megegyezik az azonos típusú változó tárolásával. A tömbelemek oszlopfolytonos elrendezésben követik egymást, tehát először a nulladik oszlop, majd az első, a második stb. Jelen példánkban tehát először (0,0), majd (1,0),(0,1),(1,1),(0,2),(1,2).

\$31–\$32 Szabad tárterület kezdete. Ez a kétbyte-os mutató mutat a tömbváltozók után kezdődő szabad tárterület első tárcímére.

\$33–\$34 Füzérváltozóterület vége, ill. szabad tárterület vége mutató. Pontosabban az utolsó szabad tárcímre mutat. Azért lehetséges, hogy mindkét területnek itt van vége, mert a füzérek a BASIC munkaterület végétől visszafelé haladva kerülnek tárolásra. A változóterületen lévő füzérdefinícóban szereplő mutató erre a területre mutat, mégpedig az aktuális füzér kezdőcímére, a hosszmutató pedig ettől a címtől visszafelé haladva a füzér hosszát adja meg. Füzérműveletek elvégzésekor, pl. A\$ = A\$ + B\$, nem módosítja A\$ értékét B\$ értékével, hanem az eredeti A\$ és B\$ után ábrázolja a kettő összegét. Ezután módosítja a változóterületen lévő A\$ mutatóját az új értékre. Így sok füzérművelet elvégzésekor nagyon hamar betelik a szabad tár, azaz összeér a tömbök és a füzérek területe. A felesleges füzéradatokat, tehát amire nem mutat mutató, el lehet dobni, így összeszűkül a füzérek területe, felszabadul a tár egy része. A füzérterület ezen aktualizálását nevezik "szemétgyűjtésnek". Programfutás közben erre automatikusan sor kerül, de a BASIC-ből mi is kiválthatjuk, ha lekérdezzük a szabad tárterületet a FRE(0) utasítással. Ez ugyanis meghívja ezt a "szemétgyűjtő" rutint. A gépi kódból a GARBAGE COLLECTION rutint kell meghívni a \$A954 címen.

\$37–\$38 A BASIC munkaterület vége mutató. Ez adja meg a BASIC által igénybe vehető legmagasabb tárcímet. Alapállapotban ez a lehető legmagasabban

van, C 16-osnál $4000-on, a Plus/4-esnél $FD00-on. Ha ezt lejjebb állítjuk, csökken a szabad tárterület.

Kitörölt BASIC program helyreállítása

Ha véletlenül kiadtunk egy NEW parancsot vagy megnyomtuk a RESET gombot, akkor az elveszett BASIC programot a következő kis gépi kódú programmal helyreállíthatjuk.

```
. 0620  A9  10      LDA  #$10   ;első sor lánckezdet mutató
. 0622  8D  02  10  STA  $1002  ;felső byte beállítása
. 0625  20  18  88  JSR  $8818  ;újraláncoló ROM rutin
. 0628  A5  22      LDA  $22    ;további mutatók beállítása
. 062A  18          CLC
. 062B  69  02      ADC  #$02
. 062D  85  2D      STA  $2D
. 062F  85  2F      STA  $2F
. 0631  85  31      STA  $31
. 0633  A5  23      LDA  $23
. 0635  69  00      ADC  #$00
. 0637  85  2E      STA  $2E
. 0639  85  30      STA  $30
. 063B  85  32      STA  $32
. 063D  60          RTS
```

A program kezdetén be kell állítani az első programsor láncmutatójának a felső byte-ját egy nem 0 értékre, mert az újraláncoló rutin másképpen nem indul el. A rutin lefutása után a $22–$23-ban a program vége - 2 érték van. Ennek figyelembevételével (ADC #$02) kell beállítani a többi mutató értékét.

2.9. ROM rutinok jegyzéke

Rengeteg energiát takaríthatunk meg, ha ahelyett, hogy minden feladatra magunk írnánk programot, inkább beépítjük a ROM éppen szükséges szubrutinját. A legtöbb periféria kezelése például komoly hardver ismeretet igényel, megtanulása helyett egyszerűbb a KERNAL rutinok kezelését elsajátítani. Olyan szubrutinok is vannak a ROM-ban, melyek megírására a leggyakorlottabb programozók sem szívesen vállalkoznának. Tanulságos lehet emellett néhány jó programozói fogás, programszervezési módszer elsajátítása is.

2.9.1. BASIC UTASÍTÁSOK BELÉPÉSI PONTJAI

HEX.	DEC.	UTASÍTÁS
$8CDA	36058	END
$ADCA	44490	FOR
$9294	37524	NEXT
$8DB0	36272	DATA
$90EE	37102	INPUT#
$9108	37128	INPUT
$969B	38555	DIM
$914F	37199	READ
$8E7C	36476	LET
$8D4D	36173	GOTO
$8EBC	35772	RUN
$8DE1	36321	IF
$8C9A	35994	RESTORE
$8D2C	36140	GOSUB
$8D83	36227	RETURN
$8E0B	36363	REM
$8CD8	36056	STOP
$8E1B	36379	ON
$9E6A	40554	WAIT
$A7F3	42995	LOAD
$A7DE	42974	SAVE
$A7F0	42992	VERIFY
$9A9D	39581	DEF
$9E12	40466	POKE
$8FE0	36832	PRINT#
$9000	36864	PRINT
$8D03	36099	CONT
$8AFF	35583	LIST
$8A98	35480	CLR
$8FE6	36838	CMD
$A7B5	42933	SYS
$A84D	43085	OPEN
$A85A	43098	CLOSE
$90B8	37048	GET
$8A79	35449	NEW
$8E0B	36363	ELSE
$B440	46144	RESUME
$B42B	46123	TRAP
$B652	46674	TRON
$B655	46677	TROFF
$B849	47177	SOUND
$B8BD	47293	VOL

HEX.	DEC.	UTASÍTÁS
$B6CD	46797	AUTO
$B544	46404	PUDEF
$C5C3	50627	GRAPHIC
$B8D1	47313	PAINT
$B9D4	47572	CHAR
$BAE2	47842	BOX
$C01E	49182	CIRCLE
$BD35	48437	GSHAPE
$BE29	48681	SSHAPE
$C4D9	50393	DRAW
$C50F	50447	LOCATE
$C51A	50458	COLOR
$C567	50535	SCNCLR
$C5B8	50616	SCALE
$B6E8	46824	HELP
$B557	46423	DO
$B603	46595	LOOP
$B5AC	46508	EXIT
$C8BC	51388	DIRECTORY
$C941	51521	DSAVE
$C951	51537	DLOAD
$C968	51560	HEADER
$C99C	51612	SCRATCH
$C9CC	51660	COLLECT
$C9DA	51674	COPY
$C9F4	51700	RENAME
$CA00	51712	BACKUP
$AE5A	44634	DELETE
$AB8F	43919	RENUMBER
$B729	46889	KEY
$FF52	65362	MONITOR

A BASIC utasítások rutinjait használni csak nagyon korlátozottan, megérteni pedig egyáltalán nem lehet a CHRGET rutin működésének ismerete nélkül. A CHRGET a RAM területen, a $0473-as címen kezdődik, bekapcsoláskor ide másolódik. Szerepe az, hogy beolvas egy byte-ot az aktuális BASIC utasításból (mindig RAM-ból), és a feltételregisztert a beolvasott értéktől függően, a BASIC osztályozási szempontjai szerint állítja.

Listája a következő:

```
. 0473   E6  3B       INC  $3B       ;CHRGET belépési pont
. 0475   D0  02       BNE  $0479
. 0477   E6  3C       INC  $3C
. 0479   78           SEI            ;CHRGOT belépési pont
. 047A   8D  3F  FF   STA  $FF3F
. 047D   A0  00       LDY  #$00
. 047F   B1  3B       LDA  ($3B),Y
. 0481   8D  3E  FF   STA  $FF3E
. 0484   58           CLI
. 0485   C9  3A       CMP  #$3A      ;kettőspont
. 0487   B0  0A       BCS  $0493
. 0489   C9  20       CMP  #$20      ;szóköz átlépés
. 048B   F0  E6       BEQ  $0473
. 048D   38           SEC
. 048E   E9  30       SBC  #$30
. 0490   38           SEC
. 0491   E9  D0       SBC  #$D0
. 0493   60           RTS
```

Ennek a rutinnak két belépési pontja van: a $0473 (CHRGET) és a $0479 (CHRGOT). A CHRGET induláskor növeli a $3B–$3C mutatót, ily módon a BASIC szöveg következő byte-jával tér vissza. A CHRGOT a mutató növelését átugorja, tehát az utasítás aktuális byte-ját lehet mégegyszer megvizsgálni vele.

Indulás után RAM-ra lapozunk, mert BASIC programot csak oda lehet írni. Utána összehasonlítások következnek, melyek az alábbiak szerint állítják a jelzőbiteket:

Jelzőbit	Beolvasott érték	
Z=1	$00	(sorvég)
Z=1	$3A	(kettőspont)
C=0	$30–$39	(számjegy)
N=1	$BA–$FF	

A BASIC utasítások végrehajtásának a menete

Az utasítás tokenje alapján az ugrási táblázatból a belépési pont címe (cím−1) a verembe töltődik, majd JMP CHRGET következik. Ennek a végén levő RTS hatására a belépési pontra kerül, hiszen ez volt a veremben. Az utasítások rutinjai tehát úgy kezdenek futni, hogy a CHRGET mutató ($3B) és a regiszterek már az utasítás utáni karakternek megfelelő értékűek. Sok BASIC rutin első utasítása már a feltételes elágazás, a paraméterek vizsgálata. PL. a BEQ itt azt jelenti, hogy ugrás, ha nincs paraméter, tehát az utasítást kettőspont vagy sor vége követi (pl. RUN-nál).

2.9.2. BASIC RUTINOK

8000	JMP $8019
	BASIC hidegstart: BASIC ROM indítása (pl. RESET)
8003	JMP $800A
	BASIC melegstart (pl. hiba után)
800A	Melegstart
8019	Hidegstart
802E	BASIC RAM inicializálás: RAM olvasó rutinok, mutatók beírása
80C2	BASIC bejelentkezés + NEW
80E5	Szabad BASIC tárkapacitás kiírása:
	XXXXX BASIC BYTES FREE
8117	BASIC vektorok írása (átmásolása)
8123	CHRGET (átmásolásra kerül a $0473-tól)
8144	Általános (önfelíró) RAM olvasó rutin átmásolódik $0494-be
8155	A $0494 RAM olvasó rutin meghívásai különböző indirekt címekkel
8653	Hibaüzenetszöveg-mutató ($24) beállítása a hibaüzenet-táblázat
	megfelelő értékére AC = hiba kód
866F	READY üzenet kiírása
867E	BASIC indítás: READY módba lépés hiba esetén: hibaüzenet + READY
	nincs hiba: READY
870F	(8712) Parancs (új BASIC sor) beolvasás
873B	Programsor törlése
8785	Programsor beszúrása
8818	Programsorok újraláncolása: az összes sor mutatójának számolása és írása
885A	Beolvasás a BASIC input pufferbe RETURN billentyű lenyomásáig
8871	BASIC verem kezelése (pl. FOR-NEXT)
88C0	RAM-blokk átmásolása
	eleje: ($5F)
	vége : ($5A)
	új vége:($58)
	Az utolsót nem másolja át!
8905	BASIC verem szabad hely vizsgálat
8923	Füzér szabad hely vizsgálat
8953	(8956) Szöveg tokenizálása
89D4	USR token
89EA	Tokenizálás utáni összehúzás
8A03	Token azonosítása 818E táblázat alapján
8A3D	BASIC sor megkeresése sorszám alapján a keresett sorszám
	14/15-ben visszajelzés: Z = 0 nincs
8A79	NEW (Z = 1)
8A98	CLR (Z = 1)
8AF1	CHRGET mutató a BASIC elejére (1000-ra)
8AFF	LIST

8B40	Egy sor kiírása (LIST)
8B6B	(8B6E) Token listázása
8BBC	RUN
8BC4	RUN sorszámmal
8BD3	(8BD6) BASIC utasítás végrehajtó ciklus
8C1A	CONT folytatási cím tárolása (025B-025C)
8C25	Token értelmezése, végrehajtás
8C6B	MID$ nem függvényként (értékadás bal oldalán)
8C74	GO
8C82	USR
8C93	Kettőspont vizsgálat (ha nem, hiba)
8C9A	RESTORE
8CC0	STOP billentyű figyelés
8CD8	STOP
8CDA	END
8D03	CONT
8D2C	GOSUB
8D4D	GOTO
8D83	RETURN
8DB0	DATA
8DBE	Kettőspont vagy sorvég keresése (pl. DATA)
8DC1	Sorvég keresése (pl. REM)
8DE1	IF
8E0B	REM (vagy nem teljesülő ELSE)
8E10	ELSE (csak teljesüléskor)
8E1B	ON
8E3E	Sorszám beolvasása és beírása egészként 14/15-be
8E7C	LET (értékadás)
8EB0	Karakteres változó értékadása
8EF4	Számjegy beolvasása, ellenőrzése és beírása FAC-ba és ARG-ba
8F04	Szöveg értékadás
8FE0	PRINT#
8FE6	CMD
9000	PRINT
903E	Új sor jel kiírása: egységszám > 127 :0D (13) + 0A (10) egységszám < 127 :0D (13)
904F	Szóköz kiírása "PRINT ,"-nek megfelelően
905F	C = 0: SPC; C = 1:TAB
9088	Szövegkiírás PRINT-nél
90A6	Szóközkiírás
90AD	Kurzor jobbra kiírása
90B0	Kérdőjel kiírása (INPUT)
90B8	GET
90EE	INPUT#

9108	INPUT
914F	READ
9208	?REDO FRROM START kiírása (INPUT)
9280	?EXTRA IGNORED kiírása (INPUT)
9294	NEXT
9314	FRMNUM: numerikus kifejezés kiértékelése
931A	Kifejezés típus ellenőrzés
9324	TYPE MISMATCH ERROR kiírása
9327	FORMULA TOO COMPLEX ERROR kiírása
932C	FRMEVL: tetszőleges kifejezés kiértékelése
9414	(9417) Kifejezés egy elemének beolvasása
9465	NOT
9477	FN
948B	")" vizsgálat, ha nem, SYNTAX ERROR
948E	"(" vizsgálat, ha nem, SYNTAX ERROR
9491	"," vizsgálat, ha nem, SYNTAX ERROR
94AD	Változó azonosítása, kiértékelése
94E8	DS$ kiértékelése
94FA	Lemezegység-hibacsatorna olvasása. Csak új lemezművelet után olvas.
9501	Numerikus változó kiértékelése
953F	ST azonosítása, beolvasása a FAC-ba
9547	ST beolvasása a FAC-ba
954D	DS azonosítása, beolvasása a FAC-ba
9555	DS$ beolvasása
9577	ER, EL azonosítása, beolvasása a FAC-ba
9583	EL beolvasása a FAC-ba
958C	ER beolvasása a FAC-ba
9599	Függvény kiértékelése
95B4	Karakteres függvény kiértékelése
95D9	Numerikus függvény kiértékelése
95F8	OR
95FB	AND
9628	Relációk vizsgálata
9640	Relációk vizsgálata szöveges típusnál
969B	DIM
96A5	Változónév beolvasása, típusazonosítás, keresés. név: \$45–\$46-ba típus: \$0D–\$0E-be kerül megtalált változó mutatója: \$47–\$48
973A	Betűvizsgálat: C = 0 nem betű.
9744	Új változó tárolása.
985B	Tömbváltozó-adatterület kezdetének számítása
989B	Tömbelem beolvasása

993A	Tömbváltozó tárolása
99C3	Tömbelem megkeresése
99F1	Tömbelemcím kiszámítása
9A2F	Szorzórutin cím számításhoz
9A62	FRE
9A7D	POS
9A86	Parancsmód vizsgálata
9A9D	DEF
9ACB	FN ellenőrzése
9B54	Helyfoglalás füzéreknek
9B66	STR$
9B74	Szöveg beolvasása
9BB0	Szövegmutató állítása a szövegverembe
9BDA	Füzérek összeadása
9C1B	Füzérek átmásolása
9C48	Füzérek beolvasása (FRESTR): AC = hossz
9CAA	Szövegmutató törlése a veremből
9CBB	CHR$
9CCF	LEFT$
9D03	RIGHT$
9D15	MID$
9D46	Füzér és paraméter beolvasása
9D61	LEN
9D67	Füzér paraméterek beolvasása
9D70	ASC
9D81	Egybyte-os egész XR-be olvasása
9D84	Egybyte-os egész XR-be olvasása
9D93	VAL
9DD2	Cím, és egybyte-os adat beolvasása (cím,adat :l. POKE) cím: $14–$15-be adat: XR-be
9DDE	Cím beolvasása $14–$15-be
9DFA	PEEK
9E12	POKE
9E1B	DEC
9E6A	WAIT
A01E	LOG
A2BE	SGN
A2DD	ABS
A358	INT
A453	"IN" és sorszám kiírása
A45F	A/X kétbyte-os pozitív egész kiírása
A5E4	SQR
A660	EXP

A707	RND
A760	BASIC verem 7C-r+1 3D-re
A769	BASIC verem 3D-ről 7C-re
A772	BASIC verem növelés Y-nál
A77D	BREAK, hiba kiírása
A785	OPEN
A78B	BASIC CHROUT
A791	BASIC CHRIN
A797	BASIC CHKOUT
A7A6	BASIC CHKIN
A7AF	BASIC GETIN
A7B5	SYS: regiszterek induláskor és visszatéréskor
	AC = 07F2
	XR = 07F3
	XR = 07F4
A7DE	SAVE
A7F0	VERIFY
A7F3	LOAD
A84D	OPEN
A85A	CLOSE
A86B	LOAD, SAVE paraméterek beállítása (SETLFS, SETNAM)
A897	","-ellenőrzés és paraméterbeolvasás X-be
A89D	Végjel (":" vagy 0) esetén visszatérési cím kivétele a veremből
A8A5	Paraméter létezésének ellenőrzése
A8Bá	OPEN paraméterek beolvasása
A8F8	DS, DS$ érvénytelenítése
A906	Füzér helyfoglalás
A954	Füzér "garbage collection" (szemétgyűjtés)
AA70	COS
AA77	SIN
AAC0	TAN
AB1A	ATN
AB8F	RENUMBER
	03/04 kezdősor
	05/06 növekmény
ADCA	FOR
AE5A	DELETE
AECA	Sorszámtartomány beolvasása (LIST)
AEF7	USING
B1ED	Szám formázása outputra
B2B7	Formátumsztring feldolgozása
B386	INSTR
B42B	TRAP
B440	RESUME
B4BE	ER$

B507	HEX$
B544	PUDEF
B557	DO
B5AC	EXIT
B603	LOOP
B652	TRON
B655	TROFF
B65B	MID$
B6CD	AUTO
B6E8	HELP
B70C	Hibás sor kiírása villogással
B729	KEY
B7A7	KEY újradefiniálással
B849	SOUND
B8BD	VOL
B8D1	PAINT
B9D4	CHAR
BA7F	Egy karakter kiírása a grafikus képernyőre
BAE2	BOX
BC56	Szögfüggvényszámítás interpolációval grafikához
BD35	GSHAPE
BE29	SSHAPE
BF79	RGR
BF85	RCLR
BF87	RLUM
BFC1	JOY
BFFD	RDOT
C01E	CIRCLE
C0D5	Egyenes rajzolása a bittérképre
C1A5	Pont rajzolása a bittérképre
C1F3	Pont vizsgálata
C21A	Karakterhely színének beállítása a bittérképnél
C291	Pont címének kiszámítása
C2AD	Karakteres koordináták kiszámítása
C2D3	SCALE
C2F6	Koordináták összeadása
C305	Koordináták kivonása
C37B	Grafikus kurzor koordináták aktualizálása
C38F	Kétbyte-os egész beolvasása
C3A5	Egybyte-os egész beolvasása
C3B6	Színtípuskód beolvasása X-be (pl. CIRCLE)
C3D9	X, Y koordináták beolvasása (pl. CIRCLE)
C48F	Egy koordináta beolvasása
C4D9	DRAW
C50F	LOCATE

C51A	COLOR
C567	SCNCLR
C5B8	SCALE
C5C3	GRAPHIC
C63C	Bittérkép kiiktatása a BASIC területből BASIC vége lejjebb tolása (C 16-os) terület felmásolása 4000 fölé (Plus/4-es)
C710	BASIC verem címmódosítása
C738	Grafikus terület felszabadítása (GRAPHIC CLR)
C7BF	Grafikus terület lefoglaltságának ellenőrzése
C7C9	Grafikus üzemmód kikapcsolása (GRAPHIC 0)
C7F0	Tárterület átmásolása (grafikus terület miatt)
C86B	Füzér-terület mérete X-be és Y-ba
C8BC	DIRECTORY
C941	DSAVE
C951	DLOAD
C968	HEADER
C99C	SCRATCH
C9CC	COLLECT
C9DA	COPY
C9F4	RENAME
CA00	BACKUP
CA16	Lemezegység-parancs elküldése
CA3F	Lemezegység-parancs összeállítása
CB1F	Lemezegység-parancs ellenőrzése
CC51	Egységszám beolvasása
CC69	File-név beolvasása
CCAA	Paraméterkapcsoló vizsgálata
CCCF	Lemezegység-hibacsatorna olvasása (ha nincs érvényes olvasás)
CD2F	ARE YOU SURE? kiírás és várakozás "Y"-ra vagy "N"-re Z = 1 (BEQ) jelzi, ha "Y"-t nyomtunk
CD40	Várakozás "Y"-ra vagy "N"-re Z = 1 (BEQ) jelzi, ha "Y"-t nyomtunk
CD57	DS, DS$ érvénytelenítése
CD74	Akkumulátor egészként való kiírása a képernyőre
CDAB	Szerzők nevének kiírása

2.9.3 ARITMETIKAI RUTINOK

8E3E	Sorszám (pozitív egész) beolvasása \$14–\$15-be \$3B szerint
9314	FRMNUM: numerikus kifejezés kiértékelése
9414	(9417) Numerikus elem kiértékelése
9871	FAC egészként A/Y-ba konvertálása
9879	Egész tipusú kifejezés kiértékelése

9D81	Egybyte-os érték X-be olvasása
9DD2	Cím és egybyte-os adat beolvasása: (l. POKE) cím \$14–\$15-be, adat X-be
9DDE	Cím beolvasása \$14–\$15-be
9E87	FAC = ARG - FAC
9E9B	FAC = (A/Y) + FAC
9E9E	FAC = ARG + FAC
9F7B	FAC mantissza invertálása
A05C	FAC = (A/Y) * FAC (ROM-ból)
A062	FAC = FAC / 2
A066	FAC = (A/Y) + FAC (ROM-ból)
A06C	FAC = (A/Y) - FAC (ROM-ból)
A072	FAC = (A/Y) / FAC (ROM-ból)
A078	FAC = (A/Y) * FAC (RAM-ból)
A07B	FAC = ARG * FAC
A0DC	ARG = (A/Y) (ROM-ból)
A107	ARG = (A/Y) (RAM-ból)
A137	FAC és ARG kitevőjének összeadása szorzáshoz
A162	FAC = FAC * 10
A183	FAC = FAC / 10
A194	FAC = (A/Y) / FAC (RAM-ból)
A197	FAC = ARG / FAC
A21F	FAC = (A/Y) (RAM-ból)
A221	FAC = (A/Y) (ROM-ból)
A24C	FAC kiírása \$5C–\$60 területre
A24F	FAC kiírása \$57–\$5B területre
A255	FAC kiírása a RAM-ba \$49 szerint
A259	FAC kiírása a RAM-ba (X/Y) szerint
A281	FAC = ARG
A291	ARG = FAC
A2A0	FAC kerekítése
A2B0	FAC előjel vizsgálata A = 01 :pozitív A = FF :negatív
A2E0	(A/Y) és FAC összehasonlítása
A327	FAC egésszé konvertálása
A37F	ASCII → FAC konverzió
A45F	(A/X) pozitív egész kiírása a képernyőre
A46F	FAC → ASCII konverzió (\$100-tól kezdődő területre)
A5E4	FAC = SQR(ARG)
A5EB	FAC = ARG ↑ (A/Y)
A5EE	FAC = ARG ↑ FAC
A627	FAC = -FAC
A6B3	A1 * X + A2 * X ↑ 3 + A3 * X ↑ 5 polinomszámítás
A6C9	Polinomszámítás

A70E	FAC = véletlen szám (timer)
A725	FAC = véletlen szám (képlettel)
AA70	FAC = COS(FAC)
AA77	FAC = SIN(FAC)
AAC0	FAC = TAN(FAC)
AB1A	FAC = ATN(FAC)
CD74	A mint egész kiírása a képernyőre

2.9.4. KERNAL ROM RUTINJAI

CE00	Interrupt (megszakítás) rutin kezdete
CE0E	IRQ belépési pont
CE60	Osztott képernyő lekezelés IRQ-ban
CECD	Hangidőzítés lekezelése IRQ-ban
CEF0	TI növelése: ($A3–$A4–$A5 óra növelése)
CF26	RDTIM: Óra tartalmának beolvasása A/X/Y-ba
CF2D	SETTIM: Óra értékének írása A/X/Y-ból
CF66	Monitorüzenet kiírása (mutató X-ben) üzenettáblázat: CF36-CF65
CF74	RETURN-jel visszaküldése (echo)
CF8A	Színbillentyűkód olvasása 07F9 = 00: RAM 07F9 = FF: ROM
CF96	RAM/ROM olvasó rutin (A1 szerint) 07F8 = 00: ROM 07F9 = FF: RAM
CFA6	ROM-ok inicializálása
CFB3	RAM olvasó rutin, átmásolásra kerül $07D9-től
CFBF	PLAY-gomb ellenőrzése, TI aktualizálása
D834	SCREEN: képernyőformátum olvasása: X = oszlop, Y = sor
D839	PLOT: kurzorpozíció írása vagy olvasása C = 1: olvasás C = 0: írás
D84E	Video alapállapotba hozása
D888	Escape N
D88B	Képernyő törlése (CLR)
D89A	HOME: kurzor a bal felső sarokba
D8A8	Kurzor-, képernyő- és színmemória címek számítása
D8C1	Billentyűzetpufferből egy karakter kiolvasása AC = 00: üres
D8EA	Beolvasás a képernyőről RETURN-ig
D965	INPUT a képernyőről
D9BA	Idézőjel figyelése, kapcsoló ($CB) állítása

DA21	Új sor (RETURN) a képernyőre
DA3D	Sor átmásolása
DA5E	Sor beszúrása
DA89	Képernyő görgetése (scroll)
DAF7	Képernyősor törlése, sorszám az X-ben
DB11	SCNKEY: billentyűzet lekérdezése és pufferelése
DB70	Billentyűzetmátrix lekérdezése
DB7A	Billentyű dekódolása
DC49	PRINT: karakterkiírás képernyőre
DC8C	Új sor (RETURN) a képernyőre (idézőjel mód törléssel)
DC9B	Escape O
DCA7	Képernyővezérlő-karakterek kiírása ($20 alattiak)
DCF1	Kurzorszín váltás
DCFA	Kurzor jobbra
DD00	Kurzor le
DD0D	Kurzor fel
DD1C	Kurzor balra
DD27	Átváltás (SHIFT/COMMODORE) kisbetűkre (CTRL N)
DD2E	Átváltás (SHIFT/COMMODORE) letiltás (CTRL H)
DD35	Átváltás (SHIFT/COMMODORE) engedélyezés (CTRL I)
DD3E	Átváltás (SHIFT/COMMODORE) nagybetűkre: CHR$(142)
DD47	Képernyővezérlő-karakterek kiírása ($80–$9F tartomány)
DD99	DEL kiírás
DDCE	INS kiírás
DE06	ESCAPE funkciók szétválogatása (ASCII kód az akkumulátorban)
DE48	ESCAPE R
DE5E	ESCAPE T
DE60	ESCAPE B
DE70	Normál képernyőablak (25x40), folytatósor törlés
DE8B	ESCAPE I
DEA0	ESCAPE D
DECB	ESCAPE Q
DEE1	ESCAPE P
DEF6	ESCAPE V
DF04	ESCAPE W
DF1D	ESCAPE L
DF20	ESCAPE M
DF26	ESCAPE C
DF29	ESCAPE A
DF2F	Kurzor alatti karakter olvasása
DF39	Folytatósor vizsgálata
DF46	Folytatósor állítása vagy törlése C = 1: 0llítás, C = 0: törlés

DF59	Folytatósor állítása
DF82	ESCAPE J
DF95	ESCAPE K
DFBF	Kurzor jobbra
DFD4	Kurzor balra
DFF6	Kurzorpozíció mentése
DFFF	Szóköz (SPACE) kiírás
E001	Képernyőkód kiírása
E011	Képernyőkód kiírása (FLASH nélkül)
E153	Soros busz TALK (beszélő)
E156	Soros busz LISTEN (hallgató)
E177	Soros busz CIOUT: byte kiküldése
E1F7	Soros busz SECOND: másodlagos cím LISTEN után
E203	Soros busz TKSA: másodlagos cím TALK után
E21D	Soros busz CIOUT: byte kiküldése puffereléssel
E22F	Soros busz UNTALK (beszélő vége)
E23D	Soros busz UNLISTEN (hallgató vége)
E252	Soros busz ACPTR: byte fogadása
E2B8	Soros busz CLOCK vonal 0-ra
E2BF	Soros busz CLOCK vonal 1-re
E2C6	Soros busz DATA vonal 0-ra
E2CD	Soros busz DATA vonal 1-re
E2D4	Soros busz DATA vonal olvasása
E2DC	1 ms késleltetés
E2EA	20 ms késleltetés
E2F8	1 ms idő a 2. időzítőbe
E2FC	20 ms idő a 2. időzítőbe
E311	Időhúzás
E319	PRESS PLAY & RECORD ON TAPE kiírása
E31B	PRESS PLAY ON TAPE kiírása
E31C	PRESS PLAY (RECORD) ON TAPE kiírása C = 1: RECORD is
E364	Kép kikapcsolás, 1. időzítő engedélyezése
E378	Visszatérés kazettakezelésből
E38D	Kazettamotor bekapcsolása
E3B0	Kazettamotor kikapcsolása
E3B7	Kazettapuffer feltöltése szóközzel
E3C3	Kazettapuffer-mutató beállítása
E3CE	Kazettakezelés megszakítása STOP miatt
E3E4	Időzítő (kazetta) IRQ kezelése
E403	18 ms idő a 2. időzítőbe
E413	Egy impulzus kiírása szalagra ideje: \$07C8–\$07C9
E447	1.00 ms idő a szalagidőzítőbe
E452	0.25 ms idő a szalagidőzítőbe
E45D	0.50 ms idő a szalagidőzítőbe

E468	0 bit kiírása szalagra
E474	1 bit kiírása szalagra
E480	Byte-kezdet jel kiírása szalagra
E48C	Egy byte kiírása szalagra
E4BA	Adatblokk kiírása szalagra
E535	Puffer kiírása szalagra
E56C	Fejléc kiírása szalagra
E5B0	Tárterület kiírása szalagra
E5F0	EOT-jel kiírása szalagra
E5FD	Egy impulzus beolvasása szalagról
E691	Egy bit beolvasása szalagról
E6D5	Byte-kezdet jel várás
E6EC	Egy byte beolvasása szalagról
E74B	Adatblokk beolvasása szalagról
E8D3	Puffer beolvasása szalagról
E8F3	Tárterület beolvasása szalagról
E91D	Blokk-kezdet keresése olvasáskor
E9CC	Szalagfejléc olvasása
EA21	Adott fejléc keresése
EA5B	RS232 megszakítás kezelése
EAF1	RS232 egy byte beolvasása
EB25	RS232 ST állítása
EB37	RS232 byte kiírása
EB46	RS232 változók törlése
EBC6	KERNAL üzenet kiírása
EBD9	GETIN
EBE8	CHRIN
EC24	Input szalagról
EC4B	CHROUT
EC8B	ACPTR
EC96	Párhuzamos IEC input
ECDF	CIOUT
ECE6	Párhuzamos IEC output
ED18	CHKIN
ED3A	Soros busz CHKIN
ED57	Szalag CHKIN
ED60	CHKOUT
ED85	Soros busz CHKOUT
EDA1	Szalag CHKOUT
EDA9	Párhuzamos IEC jelenlét ellenőrzése
EDFA	TALK
EE02	Párhuzamos IEC TALK
EE1A	TKSA
EE21	Párhuzamos IEC TKSA
EE2C	LISTEN

EE34	Párhuzamos IEC LISTEN
EE45	ATN vonal 1-re állítása
EE4D	SECOND
EE54	Párhuzamos IEC SECOND
EE5D	CLOSE
EECA	Törlés a file-táblázatból
EEE8	File-megnyitottság ellenőrzése
EEF8	File-paraméterek olvasása a táblázatból
EF08	CLALL
EF0C	CLRCHN
EF23	UNLSN
EF2A	Párhuzamos IEC UNLSN
EF3B	UNTLK
EF42	Párhuzamos IEC UNTLK
EF53	OPEN
EF8C	RS-232 OPEN
EFB8	Kazetta OPEN
F005	IEC OPEN
F043	LOAD (VERIFY)
F0F0	Szalag LOAD
F160	SEARCHING FOR "filenév" kiírása
F189	LOADING vagy VERIFYING kiírása (93 dönti el)
F194	SAVE
F1B5	IEC SAVE
F211	IEC CLOSE
F228	SAVING kiírása
F234	Szalag SAVE
F265	STOP figyelése
F273	I/O hibaüzenetek kiírása belépési pontok F273: TOO MANY FILES F276: FILE OPEN F279: FILE NOT OPEN F27C: FILE NOT FOUND F27F: DEVICE NOT PRESENT F282: NOT INPUT FILE F285: NOT OUTPUT FILE F288: MISSING FILE NAME F28B: ILLEGAL DEVICE NUMBER
F2A4	RESET rutin
F2CE	RESTOR
F2D3	VECTOR
F30B	IOINIT

F352	RAMTAS
F40C	SETNAM
F413	SETLFS
F41A	SETMSG
F41C	READST
F423	SETTMO
F427	MEMTOP
F436	MEMBOT
F445	MONITOR
F44C	MONITOR BREAK
F478	Monitor R
F4D7	Monitor M
F50A	Monitor ;
F529	Monitor >
F54B	Monitor G
F59A	8 tárcím tartalmának kiírása (>)
F5CE	Monitor C
F5D1	Monitor T
F60E	Monitor H
F66E	Monitor L, S, V
F70A	Monitor F
F724	Monitor D
F752	Egy sor disassemblálása
F7BC	Ugró utasítás címének kiszámítása
F7D4	Utasításkód elemzése
F81B	Utasítás mnemonik outputra
F91F	Monitor A
FA7D	Hexadecimális számjegy ellenőrzése
FA8B	Kétjegyű hexadecimális szám olvasása
FAA0	Hexadecimális számjegy számértékké alakítása
FAAB	Hexadecimális paraméter beolvasása
FAFB	Hexadecimális cím kiírása
FAFF	A/X kiírása hexadecimálisan + SPACE
FB05	AC kiírása hexadecimálisan + SPACE
FB08	SPACE kiírása
FB0B	? kiírása
FB10	AC kiírása hexadecimálisan
FB20	Konverzió hexadecimálisra AC-ból
FB35	Kurzor fel kiírása
FB3A	RETURN kiírása
FB3F	Olvasás az INPUT pufferből
FB5B	Monitor cím áttöltés beolvasáshoz
FB72	Monitor számláló csökkentése
FB86	Monitor végcím csökkentése

FB94	Monitor aktuális cím növelése
FBA0	Monitor címtartomány beolvasása
FBB7	A, X, Y átmeneti mentése $0110–$0112-be
FBC1	A, X, Y visszatöltése $0110–$0112-ből
FBCB	Monitor STOP-figyelés
FBD8	PRIMM: a hívó JSR utáni szöveg kiírása
FC19	IOBASE
FC1E	ROM-ok jelenlétének vizsgálata
	CBM azonosítójú, nem 1-es kódú ROM-ok indítása
FC59	ROM-ok inicializálása (FŐNIX)
FC7F	Egy byte olvasása másik ROM-ból (LONG FETCH)
	ROM BANK kód X-ben
FC89	Szubrutinhívás másik ROM-ból (LONG JUMP)
FCB3	Megszakítás (IRQ) kezdete
FCC9	ROM-térkép átkapcsolása

2.9.5. A ROM RUTINOK ÁLTAL HASZNÁLT KONSTANSOK ÉS TÁBLÁZATOK

80CC–80E4	Bekapcsolási üzenet
8105–8116	BASIC vektorok alapértéke
818E–8382	BASIC kulcsszavak
8383–8414	BASIC utasítások belépési pontjai (–1)
8415–8452	BASIC függvények belépési pontjai
8453–8470	Matematikai műveletek: precedencia és cím
8471–8652	BASIC hibaüzenetek
9439–943D	PI konstans
9FF0–A01D	LOG polinom konstansai
A444–A452	Lebegőpontos-karakteres konverzió konstansai
A5A8–A5CB	Decimálisan kerek számok
A5CC–A5E3	Óra, perc másodperc átalakítás konstansai
A632–A65F	EXP konstansok
A637–A65F	EXP polinom táblázata
A6FD–A706	RND konstansai
AAEC–AB19	Trigonometriai konstansok
AAFB–AB19	SIN polinom táblázata
AB4A–AB86	ATN polinom táblázata
AB87–AB8E	Sorszámra hivatkozó utasítások tokenjei RENUMBER-hez
BC20–BC35	Alkotók neve
BFF0–BFFA	JOY iránykód táblázata
BFFB–BFFC	JOY lekérdező maszk
C289–C290	Bittérkép pont kiírása, bitmaszk
C4AF–C4B2	Szín bitminta többszínű üzemmódhoz

C4B3–C4C6	10 db kétbyte-os szám: SIN értékek 0°-tól 90°-ig (grafikához)
C4C7–C4D8	9 db kétbyte-os szám: SIN interpolációs értékek 10°-onként
C637–C63B	Grafikus módok kódjai
CAF5–CB1E	Lemezegység-parancsok szintaxis táblázata
CD32–CD3F	ARE YOU SURE? szöveg
CD89–CDAA	Szerzők neve, kódolva
CDBA–CDFF	$FF-fel feltöltött terület
CEEE–CEEF	Hang kikapcsolása AND értékek
CF36–CF65	Monitorüzenetek
D000–D3FF	Nagybetűs kararakterkészlet
D400–D7FF	Kisbetűs karakterkészlet
D802–D833	Képernyő memória sorkezdetek (0–24) címei: D802–D81A alsó byte-ok D81B–D833 felső byte-ok
DC41–DC48	Funkcióbillentyűk ASCII kódjai
DE1A–DE47	Escape funkciók belépési pontjai
DF7A–DF81	Bitmaszk folytatósor kezeléshez
E01E–E025	Billentyűzet dekódertáblázatok kezdőcímei
E026–E066	Normál billentyűk dekódoló táblázata
E067–E0A7	Shiftelt billentyűk dekódoló táblázata
E0A8–E0E8	Commodore-os billentyűk dekódoló táblázata
E0E9–E129	Controll-os billentyűk dekódoló táblázata
E12A–E132	Szöveg shift/stop-hoz: dL"* + RUN
E133–E142	Színbillentyűk ASCII-kódjai
E143–E152	Színbillentyűkhöz rendelt színkódok
E3A2–E3AF	C1984 COMMODORE szöveg: copyright
EB58–EBC5	I/O üzenetek
F2EB–F30A	KERNAL vektorok alapértéke
F338–F351	TED kiindulási értékek
F3D2–F3D9	Funkcióbillentyű-szöveghossz alapértékek
FűDA–F40B	Funkcióbillentyű-szöveg alapértékek
F580–F599	Monitorparancsok belépési pontjai (–1)
F83D–F91E	Assembler/disassembler táblázat

2.9.6. UGRÓTÁBLÁK

Lapozórutinok ugrótáblája

```
. FCF1   4C C9 FC   JMP   $FCC9   ;Vezérlésátadás másik ROM
                                   modulba
. FCF4   4C 59 FC   JMP   $FC59   ;FŐNIX rutin: ROM modul
                                   inicializálása
. FCF7   4C 7F FC   JMP   $FC7F   ;LONG FETCH rutin: olvasás
                                   ($BE),Y szerint másik ROM-ból
. FCFA   4C 89 FC   JMP   $FC89   ;LONG JUMP rutin: szubrutinhívás
                                   ($05F0) szerint másik ROM-ból
. FCFD   4C B8 FC   JMP   $FCB8   ;LONG IRQ rutin: megszakítás
                                   hívása másik ROM-ból
```

Kiegészítő KERNAL rutinok ugrótáblája

```
. FF49   4C C2 B7   JMP   $B7C2   ;KEY
. FF4C   4C 49 DC   JMP   $DC49   ;PRINT
. FF4F   4C D8 FB   JMP   $FBD8   ;PRIMM
. FF52   4C 45 F4   JMP   $F445   ;ENTRY
```

KERNAL rutinok ugrótáblája

```
. FF81   4C 4E D8   JMP   $D84E    ;CINT
. FF84   4C 0B F3   JMP   $F30B    ;IOINIT
. FF87   4C 52 F3   JMP   $F352    ;RAMTAS
. FF8A   4C CE F2   JMP   $F2CE    ;RESTOR
. FF8D   4C D3 F2   JMP   $F2D3    ;VECTOR
. FF90   4C 1A F4   JMP   $F41A    ;SETMSG
. FF93   4C 4D EE   JMP   $EE4D    ;SECOND
. FF96   4C 1A EE   JMP   $EE1A    ;TKSA
. FF99   4C 27 F4   JMP   $F427    ;MEMTOP
. FF9C   4C 36 F4   JMP   $F436    ;MEMBOT
. FF9F   4C 11 DB   JMP   $DB11    ;SCNKEY
. FFA2   4C 23 F4   JMP   $F423    ;SETTMO
. FFA5   4C 8B EC   JMP   $EC8B    ;ACPTR
. FFA8   4C DF EC   JMP   $ECDF    ;CIOUT
. FFAB   4C 3B EF   JMP   $EF3B    ;UNTLK
. FFAE   4C 23 EF   JMP   $EF23    ;UNLSN
. FFB1   4C 2C EE   JMP   $EE2C    ;LISTEN
. FFB4   4C FA ED   JMP   $EDFA    ;TALK
. FFB7   4C 1C F4   JMP   $F41C    ;READST
. FFBA   4C 13 F4   JMP   $F413    ;SETLFS
. FFBD   4C 0C F4   JMP   $F40C    ;SETNAM
. FFC0   6C 18 03   JMP   ($0318   ;OPEN
```

```
. FFC3   6C 1A 03   JMP   ($031A)   ;CLOSE
. FFC6   6C 1C 03   JMP   ($031C)   ;CHKIN
. FFC9   6C 1E 03   JMP   ($031E)   ;CHKOUT
. FFCC   6C 20 03   JMP   ($0320)   ;CLRCHN
. FFCF   6C 22 03   JMP   ($0322)   ;CHRIN
. FFD2   6C 24 03   JMP   ($0324)   ;CHROUT
. FFD5   4C 43 F0   JMP   $F043     ;LOAD
. FFD8   4C 94 F1   JMP   $F194     ;SAVE
. FFDB   4C 2D CF   JMP   $CF2D     ;SETTIM
. FFDE   4C 26 CF   JMP   $CF26     ;RDTIM
. FFE1   6C 26 03   JMP   ($0326)   ;STOP
. FFE4   6C 28 03   JMP   ($0328)   ;GETIN
. FFE7   6C 2A 03   JMP   ($032A)   ;CLALL
. FFEA   4C F0 CE   JMP   $CEF0     ;UDTIM
. FFED   4C 34 D8   JMP   $D834     ;SCREEN
. FFF0   4C 39 D8   JMP   $D839     ;PLOT
. FFF3   4C 19 FC   JMP   $FC19     ;IOBASE

. FFF6   8D 3E FF   STA   $FF3E     ;RESET belépés
. FFF9   4C A4 F2   JMP   $F2A4     ;

. FFFC   F6 FF            $FFF6     ;RESET vektor
. FFFE   B3 FC            $FCB3     ;IRQ,BRK vektor
```

2.10. Rendszerváltozók

0000	Processzor adatirány-regisztere 0 = input, 1 = output
0001	Processzor I/O kapu b0: soros busz DATA OUT b1: soros busz CLK OUT, kazetta írás b2: soros busz ATN OUT b3: kazettás egység motor vezérlése b4: kazetta olvasás b5: – b6: soros busz CLK IN b7: soros busz DATA IN
0002	Keresett token (run-time stack)
0003–0004	RENUMBER kezdősor
0005–0006	RENUMBER növekmény
0007–0008	Keresés munkaterülete
0009	Képernyőoszlop szám az utolsó TAB-nál

000A	LOAD/VERIFY jelző: 0 = LOAD, 1 = VERIFY
000B	Input puffer számláló/dimenziószám
000C	DIM munkaterület
000D	Adattípus: FF = füzér, 00 = numerikus
000E	Számtípus: 80 = egész, 00 = valós
000F	Adatkeresés-kapcsoló LIST idézőjel mód
0010	FN kapcsoló
0011	00 = INPUT, 40 = GET, 98 = READ
0012	összehasonlítás eredménye
0013	INPUT kijelzés (?) engedélyezése 0 = engedélyezve
0014–0015	Pozitív egész címjellegű szám. (pl. POKE, GOTO)
0016	Átmeneti füzérverem mutató
0017–0018	Utolsó füzér címe
0019–0021	Ideiglenes füzérverem
0022–0025	Segédmutató-terület
0026–002A	Szorzás munkaterülete
002B–002C	BASIC program kezdete
002D–002E	BASIC program vége, változók kezdete
002F–0030	BASIC változók vége, tömbök kezdete
0031–0032	BASIC tömbök vége, szabad terület kezdete
0033–0034	BASIC szabad terület vége, füzér-adatterület kezdete
0035–0036	Ideiglenes füzérmutató
0037–0038	BASIC által használható RAM vége, füzér-adatterület vége
0039–003A	Aktuális BASIC sor száma
003B–003C	Aktuális BASIC byte: TXTPTR: CHRGET mutató
003D–003E	Ideiglenes BASIC-mutató (CONT)
003F–0040	Aktuális DATA-sorszám
0041–0042	Aktuális DATA-elem
0043–0044	INPUT-mutató
0045–0046	Aktuális BASIC változó neve
0047–0048	Aktuális BASIC változó címe
0049–004A	Ciklusváltozó-mutató
004B–004C	Ideiglenes mutató
004D	Összehasonlító érték
004E–004F	FN definíció mutató
0050–0053	Füzérleírás
0054	JMP kód ($4C)
0055–0056	Függvény vektor
0057–0060	Numerikus függvények adatterülete

0061–0066	FAC (lebegőpontos akkumulátor)
0061	FAC exponens
0062–0065	FAC mantissza
0066	FAC előjel
0067	Polinom-kiértékelés számláló
0068	FAC túlcsordulás
0069–006E	ARG (lebegőpontos akkumulátor)
0069	ARG exponens
006A–006D	ARG mantissza
006E	ARG előjel
006F	ARG-FAC előjelösszehasonlítás eredménye
0070	Kerekítő érték FAC-hoz
0071–0072	Kazettapuffer-mutató
0073–0074	AUTO utasítás növekménye
0075	Grafikus terület lefoglaltság-jelző FF = lefoglalt (12K-nál nagyobb tárnál BASIC kezdet $4000-ra kerül)
0076	Funkcióbillentyű sorszáma (0–7)
0077	Funkcióbillentyű szöveghossza
0078	Általános munkateület
0079	DS$ hossz: 00 = érvénytelen
007A–007B	DS$ cím
007C–007D	BASIC veremmutató
007E–0080	Zene átmeneti munkaterülete
0081	Parancs/program kapcsoló
0082	Átmeneti munkaterület
0083	Grafikus mód GRAPHIC 0: $00 GRAPHIC 1: $20 GRAPHIC 2: $60 GRAPHIC 3: $A0 GRAPHIC 4: $E0
0084	Aktuális betűszín
0085	Segédszín 1
0086	Alap szín
0087	Maximális képernyőoszlop-szám: 40
0088	Maximális képernyősor szám: 25
0089–008B	PAINT kapcsolók
008C–008F	Átmeneti munkaterület
0090	ST: I/O rendszerváltozó
0091	STOP-billentyű kapcsoló FF = lenyomva, 7F = nincs lenyomva
0092	Átmeneti munkaterület

0093	LOAD/VERIFY kapcsoló
0094	Soros busz puffermutatója
0095	Soros busz puffere
0096	Átmeneti munkaterület
0097	Nyitott file-ok száma
0098	Alapértelmezésbeli input egység (billentyűzet = 0)
0099	Alapértelmezésbeli output egység (képernyő = 3)
009A	Rendszerüzenet-engedélyezés kapcsolója
009B–009C	SAVE kezdőcím
009D–009E	SAVE végcím+1
009F–00A2	Átmeneti munkaterület
00A3–00A5	Óra, 1/60 s (jiffy)
00A6	Soros busz munkaterülete
00A7	Szalag írás-olvasás munkaterülete
00A8	Soros busz munkaterülete
00A9	Ideiglenes színvektor
00AA	Soros busz bitszámlálója
00AB	Aktuális file-név hossza
00AC	Aktuális logikai file-szám
00AD	Aktuális másodlagos cím
00AE	Aktuális eszközszám
00AF–00B0	Aktuális file-név mutató
00B1	Hibaüzenet rutin adatterülete
00B2–00B3	SAVE kezdőcím
00B4–00B5	LOAD kezdőcím
00B6–00B7	Kazettapuffer kezdet-mutató
00B8–00B9	Átmeneti mutató
00BA–00BB	Kazettapuffer írás-mutató
00BC–00BD	PRIMM-mutató
00BE–00BF	Mutató a LONG FETCH rutinhoz
00C0–00C1	Munkaterület képernyő-scroll-hoz
00C2	Inverz kapcsoló
00C3	Sor vége INPUT-nál
00C4–00C5	Kurzorpozíció
00C6	SCNKEY munkaterület
00C7	Input típus (GET-INPUT)
00C8–00C9	Kurzor sor-kezdet címe
00CA	Kurzorpozíció soron belül
00CB	Idézőjel-mód kapcsoló
00CC	Ideiglenes oszlopszámláló
00CD	Kurzor sorszám
00CE	Munkaterület

00CF	Beszúrás-mód (INS) számláló
00D0–00D7	SPEECH modul számára fenntartott terület
00D8–00E8	Felhasználói szabad terület
00E9	Grafikus programok munkaterülete
00EA–00EB	Kurzor színmemória-mutató
00EC–00ED	Aktuális billentyűdekóder-tábla kezdete
00EE	Aktuális billentyűzetmátrix-kód
00EF	Billentyűzetpufferben lévő karakterek száma
00F0	PRINT tiltás kapcsoló: 00 = nincs tiltva Az első billentyű lenyomásáig a PRINT-et leállítja
00F1–00F2	Monitor munkaterület
00F3–00F6	Munkaterület
00F7	Kazetta első/második menet kapcsolója
00F8	Szalagfejléc típusa
00F9	Párhuzamos/soros IEC kapcsoló
00FA	X mentés STOP ellenőrzésnél
00FB	Aktuális ROM BANK kód
00FC–00FD	Küldendő karakter (RS-232)
00FE	Kurzorsor ideiglenes tároló
00FF–010F	FAC-ASCII konverzió puffer
0110–0112	Regiszterek ideiglenes mentése: A, X, Y
0113–0122	Szín-fényerő kódok színbillentyűhöz
0123–01FF	Rendszer verem szabad rész
0200–0258	BASIC, monitor input puffer
0259–025A	Előző sorszám
025B–025C	BASIC CONT cím (megszakított utasítás eleje)
025D–02AC	Lemezegység-parancs (füzér) terület
025D	Ciklusszámláló
025E–026D	File-név
026E	1. file-név hossz
026F	1. lemezegység száma
0270–0271	1. file-név cím
0272	2. file-név hossz
0273	2. lemezegység száma
0274–0275	2. file-név cím
0276	Logikai file-szám
0277	Eszközszám
0278	Másodlagos cím
0279–027A	ID
027B	DID kapcsoló
027C–02AC	Parancspuffer
02AD–02CB	Grafikus rutinok által használt koordináta és szög értékek

02AD–02AE	Aktuális X- pozíció
02AF–02B0	Aktuális Y-pozíció
02B1–02B2	Leendő X-pozíció
02B3–02B4	Leendő Y-pozíció
02C5	Szög előjele
02C6–02C7	Szög szinusza (interpolált érték)
02C8–02C9	Szög koszinusza (interpolált érték)
02CA–02CB	Szögléptetés értéke
02CC–02E3	Többszörösen definiált terület

1. USING

02CD	Kezdőérték
02CE	Végérték
02CF	Dollárkapcsoló
02D0	Vesszőkapcsoló
02D1	Számláló
02D2	Kitevő előjel
02D3	Exponens mutató
02D4	Egész rész számjegyek száma
02D5	Igazítás-kapcsoló
02D6	Tizedespont előtti pozíció
02D7	Tizedespont utáni pozíció
02D8	+/– kapcsoló
02D9	Kitevő-kapcsoló
02DA	Kapcsoló
02DB	Karakterszámláló
02DC	Előjel
02DD	Szóköz/csillag kapcsoló
02DE	Mezőkezdet-mutató
02DF	Mezőhossz
02E0	Mezővége-mutató

2. CIRCLE

02CC–02CD	Középpont X koordináta
02CE–02CF	Középpont Y koordináta
02D0–02D1	Sugár X irányban
02D2–02D3	Sugár Y irányban
02D4–02D5	Forgási szög
02D8–02D9	Ív kezdő szög
02DA–02DB	Ív vége szög
02DC–02DD	X sugár * cos (elforgatás)
02DE–02DF	Y sugár * sin (elforgatás)

02E0–02E1	X sugár * sin (elforgatás)
02E2–02E3	Y sugár * cos (elforgatás)

3.

02CF	Füzér hossz
02D0	Felülírás-mód
02D1	Füzérpozíció-számláló
02D2	Régi bittérkép byte
02D3	Új bittérkép byte
02D5–02D6	SHAPE oszlopméret
02D7–02D8	SHAPE sorméret
02D9–02DA	SHAPE ideiglenes oszlopméret
02DB–02DC	SHAPE füzér cím
02DD	Bit mutató
02DE–02E1	Átmeneti munkaterület
02E4	Karaktergenerátor-cím felső byte-ja CHAR-hoz
02E5	Átmeneti tároló GSHAPE-hoz
02E6	SCALE-kapcsoló: 00 = ki
02E7	Duplaszélesség-kapcsoló
02E8	BOX-feltöltés kapcsoló
02E9	Bitmaszk
02EB	TRACE-mód: 00 = ki, FF = be
02EF–02F1	Átmeneti grafikus terület
02F2–02F3	Valós-egész koverzió kezdet mutatója
02F4–02F5	Egész-valós koverzió kezdet mutatója
02F6–02FD	Nem használt
02FE–02FF	Lapozásos ugrás vektora
0300–0311	BASIC vektorok
0300–0301	(8686) Hibaüzenet vektor
0302–0303	(8712) Új sor beolvasása (interpreter ciklus)
0304–0305	(8956) Tokenizálás
0306–0307	(8B6E) Token listázás
0308–0309	(8BD6) Utasítás értelmezés
030A–030B	(9417) Numerikus elem (függv.) kiértékelése
030C–030D	(896A) Felhasználói utasítás tokenizálása
030E–030F	(8B88) Felhasználói token listázása
0310–0311	(8C8B) Felhasználói token végrehajtása
0312–0331	KERNAL vektorok
0312–0313	(CE42) 2. rasztermegszakítás (képernyő alja)

0314–0315	(CE0E) Megszakítás (IRQ)
0316–0317	(F44C) Megszakítás (BRK)
0318–0319	(EF53) OPEN
031A–031B	(EE5D) CLOSE
031C–031D	(ED18) CHKIN
031E–031F	(ED60) CHKOUT
0320–0321	(EF0C) CLRCHN
0322–0323	(EBE8) CHRIN
0324–0325	(EC4B) CHROUT
0326–0327	(F265) STOP
0328–0329	(EBD9) GETIN
032A–032B	(EF08) CLALL
032C–032D	(F44C) Monitor BRK
032E–032F	(F04A) LOAD
0330–0331	(F1A4) SAVE
0332–03F2	Kazettapuffer
03F3–03F4	Adatszámláló szalagra íráskor
03F5–03F6	Adatszámláló szalagról olvasáskor
03F7–0436	RS232 input puffer (64 byte)
0437–0472	Rendszer munkaterület
0473–0493	CHRGET ($0479 CHRGOT)
0494–04A1	RAM-olvasó rutin: indirekt cím AC-ban
04A2–04A4	BASIC numerikus konstans
04A5–04AF	RAM-olvasó rutin (3B),Y szerint
04B0–04BA	RAM-olvasó rutin (22),Y szerint
04BB–04C5	RAM-olvasó rutin (24),Y szerint
04C6–04D0	RAM-olvasó rutin (6F),Y szerint
04D1–04DB	RAM-olvasó rutin (5F),Y szerint
04DC–04E6	RAM-olvasó rutin (64),Y szerint
04E7	PRINT USING: szóköz
04E8	PRINT USING: vessző
04E9	PRINT USING: tizedespont
04EA	PRINT USING: dollárjel
04EB–04EE	Ideiglenes munkaterület (INSTR)
04EF	Hiba kódszám (ER)
04F0–04F1	Hiba sorszám (EL)
04F2–04F3	TRAP sorszám
04F4	TRAP, RESUME munkaterület
04F5–04F6	Hiba címe
04F7	Veremmutató TRAP-nak
04F8–04FB	DO munkaterület
04FC	1. hang időzítő alsó byte-ja
04FD	2. hang időzítő alsó byte-ja
04FE	1. hang időzítő felső byte-ja
04FF	2. hang időzítő felső byte-ja

0500	JMP kód ($4C)
0501–0502	USR-vektor (alapállapotban $991C)
0503–0507	RND utolsó érték
0508	Hideg/melegstart kapcsoló
0509–0512	Logikai file-számok táblázata
0513–051C	Egységszámok táblázata
051D–0526	Másodlagos címek (megnyitási mód) táblázata
0527–0530	Billentyüzetpuffer
0531–0532	Szabad RAM kezdőcíme (MEMBOT)
0533–0534	Szabad RAM végcíme (MEMTOP)
0535	Párhuzamos IEC időtúllépés-kapcsoló (SETTMO)
0536	File vége kapcsoló
0537–0539	Szalag mutatók
053A	Aktuális szalag-file típus
053B	Aktuális színmemória érték
053C	Villogás (FLASH) kapcsoló
053D	Nem használt
053E	Képernyőmemória-cím felső byte-ja
053F	Billentyűzetpuffer maximális hossza
0540	Billentyűismétlés kapcsolója
	00: kurzorvezérlő-billentyűk
	40: egyik sem
	80: mindegyik
0541	Billentyűismétlés számlálója
0542	Billentyűismétlési sebesség
0543	SHIFT / CTRL / C= jelző
0544	Utolsó SHIFT mód
0545–0546	Billentyűzet dekóder tábla mutató
0547	SHIFT/C= engedélyezés/letiltás
	00 = engedélyezett
	FF = letiltva
0548	SCROLL jelző
0549–054A	Képernyőszerkesztő-munkaterület
054B–0551	MONITOR munkaterület
0552–0553	MONITOR PC
0554	MONITOR SR
0555	MONITOR AC
0556	MONITOR XR
0557	MONITOR YR
0558	MONITOR SP
0559–055A	Nem használt
055B–055C	MONITOR munkaterület
055D	Funkcióbillentyű aktuális szöveg hossza
055E	Funkcióbillentyű aktuális szöveg relatív kezdete $0567-től

055F–0566	Funkcióbillentyűk szöveghossz-táblázata
0567–05E6	Funkcióbillentyű-szövegek
05E8	Párhuzamos IEC output regiszter
05EC–05EF	ROM BANK kódok
05F0–05F1	"LONG JUMP" vektor
05F2	"LONG JUMP" akkumulátor
05F3	"LONG JUMP" X regiszter
05F4	"LONG JUMP" állapotregiszter
05F5–065D	Szabad terület a ROM-ok indítóprogramjai számára
065E–06EB	Szabad terület a SPEECH modul számára
06EC–07AF	BASIC verem
07B0	Kapcsoló a szalagra íráshoz (tár vagy puffer)
07B1	Átmeneti tároló a szalag paritásszámításhoz
07B2–07B3	Átmeneti tároló a szalagfejléc írásához
07B4	Nem használt
07B5	Bitszámláló szalagolvasáshoz
07B6	Hibaszámláló szalagolvasásnál
07B7	Hiba az első olvasási menetben
07B8–07B9	Időállandó a szalaghoz
07BA–07BB	Időállandó a szalaghoz
07BC–07BD	Időállandó a szalaghoz
07BE	Veremmutató STOP billentyűvel történő megszakításhoz
07BF	Veremmutató startbithez
07C0–07C3	Blokkolvasás-paraméterek
07C4	Hibajelző olvasáshoz
07C5	Rövid-impulzus számláló fejléc olvasáskor
07C6	Hibaszámláló olvasásnál
07C7	Ideiglenes tároló VERIFY-hoz
07C8–07C9	Szalagidőzítő
07CA–07CB	Szalagidőzítő
07CC	Startbit jelző

07CD–07D8	RS232 munkaterülete

07CD	Küldendő karakter
07CE	Output regiszter szabad ? 7. bit 0 = igen
07CF	Küldendő rendszerkarakter
07D0	Küldendő rendszerkarakter van ? 7. bit 0 = nincs
07D1	Mutató az input puffer elejére
07D2	Mutató az input puffer végére
07D3	Input pufferben lévő karakterek száma
07D4	Állapotbyte

07D5	Input rutin átmeneti tároló
07D6	Helyi szünet jelző
07D7	Vezérelt szünet jelző
07D8	Megnyitás jelző: 7. bit 1 = nyitva
07D9–07E4	RAM-olvasó rutin: indirekt cím $07DF-ben
07E5	ESCAPE B: ablak alsó sor
07E6	ESCAPE T: ablak felső sor
07E7	ESCAPE B: ablak bal szélső oszlop
07E8	ESCAPE T: ablak jobb szélső oszlop
07E9	ESCAPE L/M: SCROLL engedélyezés 7. bit 1 = engedélyezett
07EA	ESCAPE-A: beszúrásüzemmód-kapcsoló 7. bit 1 = beszúrást üzemmód
07EB	Utoljára lenyomott karakter
07EC	Görgetésjelző átmeneti tároló
07ED	Aktuális betűszín
07EE–07F1	Többszörös sorok (folytatósorok) bittérképe 0 = önálló vagy kezdősor 1 = előző sorhoz kapcsolódó
07F2–07F5	Regiszterek SYS induláskor és visszatéréskor
07F2	Akkumulátor
07F3	X-regiszter
07F4	Y-regiszter
07F5	Állapotregiszter
07F6	Billentyűzet-dekódertábla mutatója
07F7	CONTROL/S – kiírás megállítása 00 = engedélyezve
07F8	MONITOR olvasáskapcsoló: 7. bit 1 = RAM 7. bit 0 = ROM
07F9	Színbillentyűkód kapcsoló 7. bit 1 = ROM $E143–$E152 7. bit 0 = RAM $0113–$0122
07FA	ROM-maszk osztott képernyőhöz a megszakításnak $FF12 állításához
07FB	Szín- és fényerőmemória helye grafikus képernyő esetén
07FC	Kazetta motor: 7. bit 1 = működik
07FD	TI aktivizálás segédszámláló PAL/NTSC miatt
07FE–07FF	Nem használt

Nagyobb szabad területek

00D0–00E8	(25)	Szabad terület 0. lapos címzéshez
0113–01E0	(206)	Verem általában nem használt területe
03F7–0436	(64)	RS232-puffer
05A4–05E6	(67)	Funkcióbillentyű-szövegek utáni terület
0567–05E6	(128)	Funkcióbillentyű-szövegek teljes területe
05F5–06EB	(247)	Bővítőprogramok használják

2.11. Input/Output műveletek programozása

A KERNAL

A KERNAL az operációs rendszer egyik igen fontos része. Segítségével valósíthatók meg az input/output műveletek, a tárkezelés, az óraidőzítés és más hasonló műveletek. A KERNAL használatával kényelmesebbé tehetjük a munkánkat, hiszen nagyon sok feladatot nem nekünk kell programozni, hanem a kész KERNAL rutint hívhatjuk meg JSR-rel. Alkalmazása azonban nagy figyelmet igényel. Egy rutin meghívása előtt minden olyan előkészítő feladatot el kell végezni, amit a rutin megkíván. Ez az előkészítés lehet egy másik KERNAL rutin meghívása vagy egy szám beírása valamelyik regiszterbe a rutin hívása előtt.

A KERNAL tartalmaz egy szabványos ugrótáblázatot ($FF81–$FFFF), amelyen keresztül az egyes rutinokat meghívhatjuk (l. 2.9. alfejezet). Ebben az ugrótáblázatban vannak felsorolva a rutinok belépési pontjai JMP-vel. Hívhatnánk ugyan közvetlenül a rutin kezdőcímét, tehát azt a címet, amit az ugrótábla JMP-je után találunk, ez azonban gyakran nem célszerű. Az ugrótábla ugyanis szabványosított, minden COMMODORE géptípuson egyforma. Az lehet, hogy az ugrótábla egy másik gépen a tár egy másik részén hívja a rutint, az ugrótábla változatlansága viszont biztosítja, hogy a KERNAL-t használó programok ezen a gépen is működjenek.

A szabványos ugrótáblán kívül van még egy nem szabványos ugrótábla is ($FF49-$FF54). Ez a régebbi gépeken (pl. C 64) még nem létező KERNAL rutinok belépési pontjait tartalmazza.

Ha egy KERNAL rutin végrehajtásában hiba történt, a hibát a C-bit (CARRY) magasra állítása jelzi és a hiba kódja az akkumulátorba kerül. A hibakezelést érdemes mindjárt a hibák keletkezésekor végrehajtani, mert később elronthatják a program futását.

A KERNAL rutinok hiba kódszámai

0 – a STOP billentyű lenyomása megszakította a program futását
1 – túl sok file-t nyitottunk meg egyszerre
2 – már nyitott file-t akarunk megnyitni
3 – nem megnyitott file-t akarunk használni

4 – az általunk keresett file-t nem találja a rendszer
5 – az egység nincs jelen
6 – a file nem input file
7 – a file nem output file
8 – hiányzik a file neve
9 – nem megengedett egységszám

A következőkben megadjuk a KERNAL rutinok részletes leírását ABC sorrendben. Ebben szerepel a rutin neve, belépési pontja, valamint a rutin által használt regiszterek neve. Az előkészítés során a megjelölt regiszterekbe kell betölteni a rutin által megkívánt adatokat, bizonyos rutinok eredménye a lefutás után itt keletkezik.

Az előkészítő rutinok azok a KERNAL rutinok, amelyeket az aktuális rutin lefuttatása előtt meg kell (lehet) hívni. Az egyes KERNAL rutinokat belépési pont szerint rendezve is feltüntettük a ROM rutinok jegyzéke című alfejezetben. Itt található még az az ugrási cím is, amely a rutin tényleges belépési pontja.

2.11.1. KERNAL RUTINOK

ACPTR

Belépési pont	:$FFA5 (dec.65445)
Előkészítő rutinok	:TALK, TKSA
Használt regiszterek	:A, X

A rutin beolvas egy byte-ot az IEC buszról, az adat az akkumulátorba kerül. Meghívása előtt a perifériát a TALK rutinnal "beszélő" állapotba kell hozni és az esetleges másodlagos címet is ki kell küldeni a TKSA segítségével. Mivel I/O műveletről van szó, az ST változó ($90) értéke állítódik.

CHKIN

Belépési pont	:$FFC6 (dec.65478)
Előkészítő rutin	:OPEN
Használt regiszterek	:A, X

Az OPEN rutin segítségével már előzőleg megnyitott file-t lehet a CHKIN rutinnal input csatornaként definiálni. Meghívása előtt a logikai file számot az X-regiszterbe kell tölteni. A rutint kötelező a CHRIN és a GETIN rutinok előtt használni, ez alól csak az az eset kivétel, ha input eszközként a billentyűzetet használjuk.

CHKOUT

Belépési pont	:$FFC9 (dec.65481)
Előkészítő rutin	:OPEN
Használt regiszterek	:A, X

Egy, az OPEN rutinnal már előzőleg megnyitott file-t lehet a CHKOUT rutinnal

output csatornaként definiálni. A rutin hívása előtt a logikai file-számot az X-regiszterbe kell tölteni. Amíg a CHKOUT rutint nem hívjuk meg, addig az output csatorna a képernyő.

CHRIN

Belépési pont	:$FFCF (dec.65487)
Előkészítő rutinok	:OPEN, CHKIN
Használt regiszterek	:A, X

A CHRIN rutinnal az akkumulátorba olvashatunk egy byte-ot arról az input csatornáról, amelyet előkészítettünk az OPEN és CHKIN rutinokkal. Ha az input csatorna a billentyűzet, akkor a képernyőn megjelenik a villogó kurzor és a BASIC-ben megszokott INPUT rutin kerül végrehajtásra. Ügyelni kell arra, hogy a CHRIN csak a RETURN billentyű lenyomásakor tér vissza az első lenyomott billentyű kódjával. Ismételt meghívásakor kerülnek beolvasásra a további karakterek, egészen addig, míg a RETURN-t ($0D) el nem értük, ami a beolvasás végét jelenti. Ha a RETURN után ismét meghívjuk, akkor a várakozási ciklus kezdődik elölről.

Amíg nem használjuk a CHKIN rutint, addig az aktuális input eszköz a billentyűzet.

CHROUT

Belépési pont	:$FFD2 (dec.65490)
Előkészítő rutinok	:OPEN, CHKOUT
Használt regiszter	:A

A rutin az előzőleg OPEN-nal már megnyitott, CHKOUT-tal outputra definiált csatornára küld egy byte-ot. Ezt a rutin hívása előtt az akkumulátorba kell tölteni. Ha meghívása előtt nem használtuk a CHKOUT rutint, a kiküldött karakter a képernyőre kerül.

CIOUT

Belépési pont	:$FFA8 (dec.65448)
Előkészítő rutinok	:LISTEN, SECOND
Használt regiszter	:A

A rutin segítségével egy byte-ot küldhetünk el az IEC buszon. Előzőleg azonban az IEC buszra csatlakoztatott eszközt a LISTEN rutinnal "hallgató" állapotba kell hozni és ha szükséges, a SECOND rutinnal ki kell adni egy másodlagos címet is. A kiküldendő byte-ot előzőleg az akkumulátorba kell tölteni, majd ezután kell a CIOUT rutint meghívni. Ez a rutin egy egybyte-os puffert használ, így minden egyes meghívásakor az előző byte kerül elküldésre. Az utolsó byte az UNLSN meghívásakor kerül az IEC buszra.

CINT

Belépési pont	:$FF81 (dec.65409)
Előkészítő rutin	:–
Használt regiszterek	:A, X, Y

A rutin inicializálja a képernyőszerkesztőt, beleértve a képernyőtörlést, az idézőjel-üzemmód megszüntetését stb.

CLALL

Belépési pont :$FFE7 (dec.65511)
Előkészítő rutin :–
Használt regiszterek :A, X

Ez a rutin lezárja az összes, előzőleg OPEN-nel megnyitott logikai file-t. A meghíváskor törlődik a nyitott file-ok táblázata, majd meghívásra kerül a CLRCHN rutin is, s hatására az I/O csatornák kezdeti (default) értéke is visszaáll. Az aktuális input eszköz a billentyűzet, aktuális output eszköz pedig a képernyő lesz.

CLOSE

Belépési pont :$FFC3 (dec.65475)
Előkészítő rutin :–
Használt regiszterek :A, X, Y

A rutin lezárja az akkumulátorba töltött logikai file-számnak megfelelő csatornát, amelynek meg kell egyezni az OPEN-ben használt logikai file-számmal. Ha az OPEN után a CHKIN vagy CHKOUT rutinnal új input vagy output csatornát definiáltunk, a CLOSE előtt meg kell hívni a CLRCHN rutint.

CLRCHN

Belépési pont :$FFCC (dec.65484)
Előkészítő rutin :–
Használt regiszterek :A, X

A CLRCHN visszaállítja az I/O csatornák kezdeti (default) értékét. Ezután az aktuális input eszköz száma 0 (billentyűzet), az aktuális output eszköz száma pedig 3 (képernyő) lesz. Ha a nyitott csatornák egyike az IEC busz volt, akkor a megfelelő UNLSN és UNTLK rutinok is végrehajtásra kerülnek.

GETIN

Belépési pont :$FFE4 (dec.65508)
Előkészítő rutinok :OPEN, CHKIN
Használt regiszterek :A, X, Y

A rutin a billentyűzetről vagy az RS232-es csatornáról egyetlen byte-ot tölt az akkumulátorba. Ha a puffer üres, az akkumulátorba 0 kerül. Ha az input csatorna az IEC busz vagy a kazettás egység, akkor a megfelelő CHRIN rutin kerül végrehajtásra.

IOBASE

Belépési pont :$FFF3 (dec.65523)
Előkészítő rutin :–
Használt regiszterek :X, Y

Az IOBASE rutin az I/O eszközök által használt tárrész kezdőcímével (jelen

esetben $FD00) tér vissza. Az X-regiszterben van az alsó, az Y-regiszterben pedig a felső byte. Ez a cím géptípusonként különbözik, az aktuális ROM kiosztás függvénye. Segítségével elvileg megoldható olyan programok írása, amely ennek felhasználásával esetleg több gépen is fut.

IOINIT

Belépési pont :$FF84 (dec.65412)
Előkészítő rutin :–
Használt regiszterek :A, X, Y

A rutin inicializálja a TED-et és valamennyi I/O eszközt.

LISTEN

Belépési pont :$FFB1 (dec.65457)
Előkészítő rutin :–
Használt regiszter :A

A LISTEN rutin az akkumulátorban lévő számnak megfelelő eszközszámú perifériát utasítja, hogy legyen "hallgató" állapotban. Ennek hatására az eszköz készen áll az adatok fogadására. Az akkumulátorba töltendő egységszámnak 0–31 közé kell esnie.

LOAD

Belépési pont :$FFD5 (dec.65493)
Előkészítő rutinok :SETLFS, SETNAM
Használt regiszterek :A, X, Y

A rutin közvetlenül a tárba tölt az input eszközről egy file-t. A rutin hívása előtt a file-ra vonatkozó adatokat a SETLFS és SETNAM rutinokkal be kell állítani. Ha a SETLFS-ben a megnyitási mód helyére 0-t írtunk, akkor a betöltés kezdőcímét nekünk kell beállítani az X- és az Y-regiszterekben. Ha a megnyitási mód értéke 1 volt, akkor a betöltés – kazettás egység esetén – a fejlécben tárolt kezdőcímtől történik. Lemezegységnél ezt az információt a file első két byte-ja tartalmazza. A LOAD rutin nemcsak betöltésre, de ellenőrzésre is használható. Ha meghívása előtt az akkumulátor tartalma 0, akkor LOAD, ha 1, akkor VERIFY történik.

MEMBOT

Belépési pont :$FF9C (dec.65436)
Előkészítő rutin :–
Használt regiszterek :X, Y

A rutinnal írhatjuk vagy olvashatjuk a "BASIC RAM kezdete" mutató értékét. Ha meghívásakor a C-bit alacsony, akkor az X- és az Y-regiszterek tartalmát a mutatóba tölti. Ha a C-bit 1, akkor a mutató értékét kiolvassa és tartalma az X- és az Y-regiszterekbe kerül.

MEMTOP

Belépési pont :$FF99 (dec.65433)
Előkészítő rutin :–

Használt regiszterek :X, Y

A rutinnal írhatjuk vagy olvashatjuk a "BASIC RAM vége" mutató értékét. Ha meghívásakor a C-bit alacsony, akkor az X- és az Y-regiszterek tartalmát a mutatóba tölti. Ha a C = 1, akkor a mutató értékét kiolvassa, tartalma az X- és az Y-regiszterekbe kerül.

OPEN

Belépési pont :$FFC0 (dec.65472)
Előkészítő rutinok :SETLFS, SETNAM
Használt regiszterek :A, X, Y

Az OPEN rutin feladata hasonló a BASIC OPEN parancshoz, egy file megnyitására szolgál. Előtte azonban a SETLFS és SETNAM rutinokkal be kell állítani a file-paramétereket, és csak ezután szabad az OPEN rutint meghívni.

PLOT

Belépési pont :$FFF0 (dec.65520)
Előkészítő rutin :–
Használt regiszterek :A, X, Y

A rutinnal a C jelzőbittől függően kiolvashatjuk, illetve beállíthatjuk a kurzor pozícióját az X- és az Y-regiszterek segítségével. Ha meghívásakor a C-bit magas, a kurzorpozíció X, Y koordinátái az X- és az Y-regiszterekbe töltődnek. Az X-regiszterbe a vízszintes, az Y-regiszterbe a függőleges koordináta kerül. Alacsony C-bit esetén a megfelelő regiszterek tartalma szerint állítódik a kurzorpozíció.

RAMTAS

Belépési pont :$FF87 (dec.65415)
Előkészítő rutin :–
Használt regiszterek :A, X, Y

A rutin teszteli a RAM-ot és a "BASIC RAM eleje" és "BASIC RAM vége" mutatókat állítja. Segítségével programból megállapíthatjuk, hogy mekkora a RAM területe (van-e bővítő). A rutin újraírja a funkcióbillentyűk szövegeit is.

Meghívni csak SEI után szabad, és mivel a teljes rendszerváltozó területet törli, meghívása után a RESTOR és CINT rutinokat is meg kell hívni.

RDTIM

Belépési pont :$FFDE (dec.65502)
Előkészítő rutin :–
Használt regiszterek :A, X, Y

Az RDTIM rutin a rendszerváltozók területén lévő óra beolvasására szolgál. Ez megegyezik a BASIC TI$ olvasásával. Az óra 3 byte-ot foglal le a $A3–$A5 területen. Az RDTIM meghívásakor az akkumulátorba töltődik a $A5 helyen lévő alsó byte értéke, az X-regiszterbe a középső, az Y-regiszterbe pedig a legfelső byte értéke kerül.

READST

Belépési pont :$FFB7 (dec.65463)
Előkészítő rutin :–
Használt regiszter :A

A rutin az akkumulátorba tölti a 0. lapon lévő, az I/O eszközök állapotát jelző cím tartalmát. Ez megegyezik a BASIC ST változó olvasásával, ami jelen esetben a $90 cím olvasását jelenti.

RESTOR

Belépési pont :$FF8A (dec.65418)
Előkészítő rutin :–
Használt regiszterek :A, X, Y

Ez a rutin visszaállítja eredeti értékükre az operációs rendszer vektorait. Ezek a vektorok a $0312-$0331-ig terjedő területet foglalják el.

SAVE

Belépési pont :$FFD8 (dec.65496)
Előkészítő rutinok :SETLFS, SETNAM
Használt regiszterek :A, X, Y

A rutin a tár egy előre megadott területét menti ki a SETLFS és SETNAM rutinokkal meghatározott file-ba. A rutin hívása előtt a kezdőcímet egy 0. lapos mutatóba, a mutató címét pedig az akkumulátorba kell tölteni. A végcímet az X- és az Y-regiszterpárba kell írni, ennek a címnek a tartalma már nem kerül a file-ba. Az X-regiszterbe a cím alsó, az Y-ba pedig a felső byte-ját kell írni.

SCNKEY

Belépési pont :$FF9F (dec.65439)
Előkészítő rutin :–
Használt regiszterek :A, X, Y

Ez a rutin a billentyűzetet kezeli, ami közönséges esetben a megszakítás feladata, alapállapotban tehát a megszakító rutin hívja meg. Használata csak akkor javasolt, ha a megszakítás le van tiltva és a billentyűzetről várunk adatot. Meghívásakor a lenyomott billentyű ASCII kódja a billentyűzetpufferbe kerül, visszatéréskor ezt az értéket az akkumulátor is tartalmazza. Ha nincs lenyomva semmi, az akkumulátor tartalma $FF.

SCREEN

Belépési pont : $FFED (dec.65517)
Előkészítő rutin : –
Használt regiszterek : X, Y

Meghívása után az X- és az Y-regiszterek tartalmazzák a képernyő sor- és oszlopszámát. Jelen esetben X = 40, Y = 25. Ez a rutin arra szolgál, hogy esetleges szoftver kompatibilitást biztosítson például egy 80 x 25-ös képernyőméretű géppel, amiben szintén szerepel ez a KERNAL rutin.

SECOND

Belépési pont : $FF93 (dec.65427)
Előkészítő rutin : LISTEN
Használt regiszter : A

Az előzőleg a LISTEN rutinnal már "hallgató" állapotba helyezett eszköznek a SECOND rutinnal küldhetünk el egy másodlagos címet, ami a megnyitási módra vonatkozik. A másodlagos címnek megfelelő parancs byte értékét előzőleg az akkumulátorba kell tölteni. Ez a rutin csak az output műveleteknél használható, a TALK rutinnal "beszélő" állapotba hozott eszköznél nem lehet alkalmazni.

SETLFS

Belépési pont : $FFBA (dec.65466)
Előkészítő rutin : –
Használt regiszterek : A, X, Y

Az I/O műveletek során használt file-paramétereket állítja be. Funkciója lényegében az OPEN utasítás három paraméterének megfelelő helyre való beírása. Az első paraméter a logikai file-szám, amit az akkumulátorba kell tölteni. A második paramétert, az egységszámot tartalmazza az X-regiszter, míg az esetleges másodlagos címet (a megnyitási módot) az Y-regiszterbe kell tölteni. Ezután hívhatjuk meg a SETLFS rutint, ami tulajdonképpen a 0. lapon lévő rendszerváltozó-terület megfelelő helyeire írja be az aktuális értékeket ($AC, $AD, $AE).

SETMSG

Belépési pont : $FF90 (dec.65424)
Előkészítő rutin : –
Használt regiszter : A

A rutin meghívásával beállíthatjuk, hogy engedélyezett-e az I/O üzenetek és I/O hibaüzenetek kiírása. Ez a rutin tulajdonképpen a $9A címen lévő rendszerváltozót állítja be, aminek csak a legfelső két bitje számít. A 7. bit az I/O üzenetek engedélyezésének kapcsolója: 0 = letiltva, 1 = engedélyezve. A 6. bit az I/O hibaüzenetek kiírását engedélyezheti: 0 = letiltva, 1 = engedélyezve. A rutin meghívása előtt az akkumulátor értékét ennek megfelelően kell beállítani. Visszatéréskor az akkumulátor az ST változó aktuális értékét tartalmazza.

SETNAM

Belépési pont : $FFBD (dec.65469)
Előkészítő rutin : –
Használt regiszterek : A, X, Y

A rutin az I/O műveletekben szereplő file-név paramétereit tölti a megfelelő helyre. Ez lehet a LOAD, a SAVE, de az OPEN után szereplő név is. Meghívása előtt az akkumulátorba kell tölteni a file-név hosszát, az X/Y-regiszterpár pedig a név kezdőcímét (X = alsó, Y = felső byte) kell, hogy tartalmazza. A rutint a KERNAL OPEN meghívása előtt kötelező alkalmazni. Ha az OPEN-ben nem szerepel file-név, akkor az akkumulátornak 0-t kell tartalmaznia.

SETTIM

Belépési pont	: $FFDB (dec.65499)
Előkészítő rutin	: –
Használt regiszterek	: A, X, Y

A rutin a 0. lapon lévő óra beállítására alkalmas. Meghívásakor az A/X/Y regiszterben lévő értékek az óra megfelelő byte-jaiba töltődnek. Az akkumulátor tartalma kerül a legmagasabb helyiértékű, az X-regiszter a középső, az Y-regiszter pedig a legalsó helyiértékű címre.

SETTMO

Belépési pont	: $FFA2 (dec.65442)
Előkészítő rutin	: -
Használt regiszter	: A

A rutin használatának csak abban az esetben van értelme, ha a géphez egy párhuzamos IEEE bővítőegységet csatlakoztattunk. A SETTMO rutin a rendszerváltozók területén, a $0535 címen lévő TIMEOUT kapcsolót állítja be, az akkumulátorba töltött értéknek megfelelően.

STOP

Belépési pont	: $FFE1 (dec.65505)
Előkészítő rutin	: –
Használt regiszterek	: A, X

A rutin ellenőrzi, hogy meghívásakor a STOP billentyű le van-e nyomva. Ha nem volt lenyomva, akkor az akkumulátor tartalma visszatéréskor $FF, a Z-jelzőbit pedig alacsony. Abban az esetben, ha a STOP billentyű le volt nyomva, akkor végrehajtásra kerül a CLRCHN rutin, valamint a billentyűzetpuffer is kiürül. Az akkumulátor tartalma visszatéréskor 0, a Z-bit magasra állítódik. Ha az UDTIM nem működik (például a megszakítás le van tiltva), akkor ez a rutin sem működik helyesen.

TALK

Belépési pont	: $FFB4 (dec.65460)
Előkészítő rutin	: –
Használt regiszter	: A

A rutin utasítja az IEC buszon lévő eszközt, hogy legyen "beszélő" állapotban. Meghívása előtt az inputként használt eszköz számát az akkumulátorba kell tölteni.

TKSA

Belépési pont	: $FF96 (dec.65430)
Előkészítő rutin	: TALK
Használt regiszter	: A

Ez a rutin a TALK rutinnal már "beszélő" állapotba helyezett eszköznek egy másodlagos címet továbbít. A másodlagos címnek (az OPEN utasítás harmadik paramétere) megfelelő parancs byte-ot meghívás előtt az akkumulátorba kell tölteni. Ügyeljünk arra, hogy a TKSA rutin csak az input műveleteknél használható, tehát a LISTEN után nem szabad használni.

UDTIM

Belépési pont	: $FFEA (dec.65514)
Előkészítő rutin	: –
Használt regiszterek	: A, X

Az UDTIM rutin eggyel növeli a 0. lapon lévő óra tartalmát, valamint ellenőrzi a STOP billentyű lenyomását is. Ehhez azonban nem a STOP KERNAL rutint hívja meg, hanem a $DB70 rutin felhasználásával a $91 címen lévő STOP kapcsolót állítja be. Alapállapotban az UDTIM rutint a megszakítási rendszer másodpercenkét 60-szor hívja meg.

UNLSN

Belépési pont	: $FFAE (dec.65454)
Előkészítő rutin	: –
Használt regiszter	: A

A rutin hívásával befejezzük az IEC buszon az adatküldést valamennyi "hallgató" állapotban lévő perifériára. Meghívásakor kerül elküldésre a CIOUT rutin által küldött utolsó byte, majd az összes "hallgató" állapotban lévő eszköz befejezi az adatok fogadását, "figyelő" állapotba kerül.

UNTLK

Belépési pont	: $FFAB (dec.65451)
Előkészítő rutin	: –
Használt regiszter	: A

A rutin hívásával befejezzük az IEC buszon az adatok fogadását a "beszélő" állapotban lévő perifériáról. Meghívása után a "beszélő" állapotban lévő eszköz befejezi az adatok küldését, "figyelő" állapotba kerül.

VECTOR

Belépési pont	: $FF8D (dec.65421)
Előkészítő rutin	: –
Használt regiszterek	: A, X, Y

A VECTOR rutinnal a C-jelzőbittől függően írhatjuk vagy olvashatjuk a rendszerváltozó területen lévő KERNAL ugrási vektorokat ($0312-$0331). Hívása előtt ki kell jelölnünk egy táblázatot, amelybe majd a kiolvasott értékek kerülnek ill. ahonnan a vektortáblázat új étrtékeit kívánjuk beírni. A táblázat kezdőcímét a meghívás előtt az X- és az Y-regiszterekbe kell tölteni. Ha a C-jelzőbit alacsony, a táblázatban szereplő értékek a rendszervektorokba töltődnek. Ha a C-bit 1, akkor a rendszervektorok tartalma a kijelölt táblázatba kerül.

2.11.2. KIEGÉSZÍTŐ KERNAL RUTINOK

KEY

Belépési pont	: $FF49 (dec.65353)
Elókészítő rutin	: –
Használt regiszterek	: A, X, Y

A rutinnal az egyes funkcióbillentyűkhöz rendelhetünk füzéreket a BASIC KEY utasításhoz hasonlóan. Meghívása előtt az akkumulátorba kell tölteni a hozzárendelendő füzér hosszát, kezdőcímét pedig a $22–$23-as mutatónak kell tartalmaznia. A funkcióbillentyű sorszámát (0-tól–7-ig) a $76-os tárcímbe kell írni.

PRINT

Belépési pont	: $FF4C (dec.65356)
Előkészítő rutin	: –
Használt regiszter	: A

A BASIC PRINT utasításhoz hasonló a funkciója. Segítségével a képernyőre írhatunk egy karaktert. A kiküldendő karakter ASCII kódját meghívás előtt az akkumulátorba kell tölteni. Visszatéréskor az A/X/Y-regiszterek értékei változatlanok. A rutin a CHROUT rutinhoz hasonlóan működik, de mindig a képernyőre ír, függetlenül attól, hogy mi az aktuális (CMD) output eszköz. Természetesen a megszokott módon kezeli a vezérlőkódokat is (pl. ESCAPE).

PRIMM

Belépési pont	: $FF4F (dec.65359)
Előkészítő rutin	: –
Használt regiszter	: –

A rutinnal karaktersorozatot küldhetünk ki az éppen aktuális output eszközre. Ezt a karaktersorozatot közvetlenül a hívó JSR $FF4F után kell elhelyezni és 00 byte-tal kell lezárni. A rutin a karaktereket egyenként küldi el a CHROUT rutin segítségével. Az adatküldés befejezésével a 00 byte utáni címen folytatódik a program futása. A küldendő karaktersorozat hossza max. 255 lehet.

ENTRY

Belépési pont	: $FF52 (dec.65362)
Előkészítő rutin	: –
Használt regiszter	: –

A gépi kódú monitorba ezen a címen keresztül léphetünk be.

2.11.3. INPUT/OUTPUT MŰVELETEKKEL KAPCSOLATOS ÁLTALÁNOS TUDNIVALÓK

Az I/O műveleteknél gyakran szükséges, hogy tájékoztatást kapjunk a végrehajtott művelet után a periféria állapotáról. A rendszerváltozók területén, a $90 címen lévő I/O állapotváltozónak ez a feladata. Értéke megegyezik a BASIC ST változó értékével. Vizsgálata döntési adatot szolgáltathat a program során feltételes elágazásokhoz. Az egyes bitek jelentése a következő:

BIT	IEC busz	Kazettás egység
0	Időtúllépés íráskor	
1	Időtúllépés olvasáskor	
2		Rövid blokk
3		Hosszú blokk
4		Olvasási hiba
5		Paritás hiba
6	File vége	File vége
7	Az egység nincs jelen	Szalag vége

Bármely perifériánál OPEN-nel történő adatcsatorna megnyitáshoz a következő paramétereket kell megadnunk:

1) Logikai file-szám
2) Periféria egységszám
3) Másodlagos cím (megnyitási mód)
4) Paraméter füzér (file-név)

Logikai file-szám

Ezen az azonosítószámon keresztül érhetjük el az eszközt (file-t). Megnyitáskor ezt a logikai file-számot rendeljük az egységhez (azon belül egy file-hoz), később ezzel a számmal hivatkozhatunk rá. Értéke 1–255 között lehet.

Egységszám

A csatlakoztatott perifériákat a C 16-os vagy a Plus/4-es egy egyedi azonosító, az egységszám alapján különbözteti meg. Értéke 0-tól 15-ig terjedhet. Az egyes egységszámok jelentése a következő:

0 – Billentyűzet
1 – Kazettás egység
2 – RS232 csatorna
3 – Képernyő
4 – Nyomtató
5 – Egyes nyomtatók átkapcsolhatók 5-ös egységszámra
8 – Lemezegység. Programból átszámozható 9-től 15-ig.

A felsorolt egységek közül a billentyűzet csak input egységként, a nyomtató csak output egységként használható. Ellenkező értelmű használatuk hibajelzést eredményez.

Megnyitási mód

Jelentése a különböző perifériáknál más és más. Itt csak azokat a perifériákat soroljuk fel, amelyeknél a megnyitási mód értelmezve van.

Kazettás egység:

szekvenciális file-ok esetén	0 –	megnyitás inputra
	1 –	megnyitás outputra EOT kiírás nélkül
	2 –	megnyitás outputra EOT kiírással
LOAD és SAVE esetén	0 –	BASIC program
	1 –	gépi kódú program

Nyomtató: a nyomtató típusától függően változó. Leggyakoribb értéke:
0 – nagybetű/grafikus mód
7 – kisbetű/nagybetű mód

Lemezegység:	0	–SAVE
	1	–LOAD
	2–14	–Fizikai adatcsatornák
	15	–Parancs- (hiba) csatorna

Paraméter füzér

Kazettás egység: a megnyitandó file neve

RS232 csatorna: az adatforgalom további paramétereit határozza meg. Hossza max. 4 karakter, ebből az első kettő kötelező.

Első karakter:
0.– 3. bit – átviteli sebesség kódja (baud)

0000	– parancs füzér utolsó két byte-ja –határozza meg.
0001	– 50
0010	– 75
0011	– 110
0100	– 134,5
0101	– 150
0110	– 300
0111	– 600
1000	–1200
1001	–1800
1010	–2400

4. bit	–	nem használt	
5.– 6. bit	–	szóhossz:	
		00	– 8 bit
		01	– 7 bit
		10	– 6 bit
		11	– 5 bit
7. bit	–	STOP-bit:	
		0	– 1 STOP bit
		1	– 2 STOP bit

Második karakter:

0. bit	–	összeköttetés típusa	
		0:	háromvonalú összeköttetés (nincs HANDSHAKE)
		1:	tetszőleges, leggyakrabban 5 vonalú összeköttetés
1.– 3. bit	–	nem használt	
4. bit	–	0:	teljes duplex (valós kétirányú összeköttetés)
		1:	félduplex (egyszerre csak egy adó lehet)
5.– 7. bit	–	paritás funkciók	
		000 –	nincs paritás bit
		001 –	páratlan paritás
		011 –	páros paritás
		101 –	a 8. bit mindig 1
		111 –	a 8. bit mindig 0

Harmadik karakter: átviteli sebesség alsó byte

Negyedik karakter: átviteli sebesség felső byte

Nyomtató: paraméter füzér nincs értelmezve

Lemezegység: LOAD/SAVE esetén a paraméter füzér a file neve. Adatcsatornák esetén (2–14) a file-név, a file-típus és a megnyitás módjára vonatkozó információ. 15-ös csatorna esetén lemez parancs kerül elküldésre.

Példák az I/O rutinok használatára

A következő részben felsorolunk néhány példát az I/O műveletekben szereplő KERNAL rutinok használatára. A legtöbb példaprogram mellé leírjuk az azonos feladat végrehajtására alkalmas BASIC programot is az analógia jobb követhetősége céljából. A példákban szereplő gépi kódú programokat vagy monitorral vagy az ASS–16 programmal írtuk meg.

A PROBA nevű szekvenciális file-t a szalagos egységen megnyitjuk outputra az OPEN utasítással:

```
OPEN 3,1,2,"PROBA"
```

Az ennek megfelelő gépi kódú program:

```
10   SYS 12224
20   .OPT P,OO
30   SETLFS     =   $FFBA
40   SETNAM     =   $FFBD
50   OPEN       =   $FFC0
60   CHKOUT     =   $FFC9
70   CHROUT     =   $FFD2
80   CLRCHN     =   $FFCC
90   CLOSE      =   $FFC3
100   *=    $2000      ;indítás:SYS 8192
110   LDA   #3         ;logikai file-szám
120   LDX   #1         ;egységszám
130   LDY   #2         ;megnyitási mód outputra+EOT
140   JSR   SETLFS
150   LDA   #5 .       ;file-név hossza
160   LDX   #<FNEV     ;file-név kezdőcíme alsó byte
170   LDY   #>FNEV     ;file-név kezdőcíme felső byte
180   JSR   SETNAM
190   JSR   OPEN
```

BASIC-ben a következőképpen lehet ebbe a megnyitott file-ba adatokat írni:

```
PRINT#3,CHR$(13);"A";
```

Gépi kódban ennek megfelelője (az előző példa folytatása):

```
200                    ;ADATKÜLDÉS
210   LDX   #3         ;logikai file-szám
220   JSR   CHKOUT     ;output csatorna definiálása
230   LDA   #13        ;újsorjel
240   JSR   CHROUT     ;kiküldése
250   LDA   #"A"       ;egy karakter
260   JSR   CHROUT     ;kiküldése
```

Az adatküldés – természetesen céljainknak megfelelően – tetszőlegesen folytatható. Ha befejeztük az adatküldést, a file-t le kell zárni.

```
CLOSE 3
```

Gépi kódú megfelelője (szintén az előző példa folytatása):

```
270                      ;CSATORNA LEZARAS
280   JSR    CLRCHN      ;I/O eszközök alapértélmezésbe
290   LDA    #3          ;logikai file-szám
```

```
300  JSR     CLOSE            ;a file lezárása
305  RTS
310  FNEV   .ASC "PROBA"      ;a file neve
320  .END
```

Ezt a programot, ha a fordítás után futtatjuk, akkor létrehoztuk a szalagos egységen a PROBA nevű szekvenciális file-t. Most próbáljuk visszaolvasni és a képernyőre kiírni a tartalmát.

```
10  OPEN 3,1,0,"PROBA"
20  GET#3,A$
30  PRINTA$
40  IF ST=0 THEN 20
50  CLOSE 3
```

Gépi kódú programmal:

```
10:SYS  12224
20:     2000                               .OPT   P,OO
30:     2000                     SETLFS    =      $FFBA
40:     2000                     SETNAM    =      $FFBD
50:     2000                     OPEN      =      $FFC0
60:     2000                     CHKIN     =      $FFC6
70:     2000                     CHRIN     =      $FFCF
80:     2000                     CLRCHN    =      $FFCC
90:     2000                     CLOSE     =      $FFC3
95:     2000                     READST    =      $FFB7
97:     2000                     PRINT     =      $FF4C
100:    2000                               *=     $2000
110:    2000   A9  03                      LDA    #3
120:    2002   A2  01                      LDX    #1
130:    2004   A0  00                      LDY    #0;megnyitás inputra
140:    2006   20  BA  FF                  JSR    SETLFS
150:    2009   A9  05                      LDA    #5
160:    200B   A2  2E                      LDX    #<FNEV
170:    200D   A0  20                      LDY    #>FNEV
180:    200F   20  BD  FF                  JSR    SETNAM
190:    2012   20  C0  FF                  JSR    OPEN
                                 ;
210:    2015   A2  03                      LDX    #3
220:    2017   20  C6  FF                  JSR    CHKIN;input csat.def.
230:    201A   20  CF  FF        OLV       JSR    CHRIN;beolvasás
240:    201D   20  4C  FF                  JSR    PRINT;kiírás
250:    2020   20  B7  FF                  JSR    READST; file vége ?
260:    2023   F0  F5                      BEQ    OLV;ha nem, olvasás
                                 ;
```

```
280:  2025  20  CC  FF          JSR   CLRCHN
290:  2028  A9  03              LDA   #3
300:  202A  20  C3  FF          JSR   CLOSE
305:  202D  60                  RTS
310:  202E  50  52  4F  FNEV    .ASC  "PROBA"
320:                            .END
```

Ezzel a programmal tetszőleges hosszúságú soros file-t olvashatunk be szalagról és írhatjuk ki képernyőre a tartalmát. A file végét az ST változó jelzi, amit a READST rutinnal figyelünk.

Ezután nézzük meg, hogyan lehet a SAVE és LOAD rutinokat használni. A másodlagos cím jelentése ebben az esetben a következő:

0 – BASIC program, a BASIC kezdet mutató által megjelölt helyre töltődik vissza.

1 – Gépi kódú program, az eredeti helyére töltődik vissza.

Lemezegység esetén ezt a címet a file első két byte-ja tartalmazza, szalagos file esetén ez az információ a fejlécben van. Kazettás file esetén a másodlagos cím 1. bitje jelzi, hogy a file végén kell-e EOT blokkot írni. 0 = nem kell, 1 = kell. Így pl. a 3-as másodlagos cím jelentése: gépi kódú program + EOT.

Az itt következő példaprogrammal a C 16-os vagy a Plus/4-es $3800-tól kezdődően áttöltött karakterkészletét írjuk szalagra.

Monitorból:

```
S "KARAKTER",1,3800,4000
```

Ugyanezt a feladatot végzi az alábbi gépi kódú program:

```
20:SYS  12224
30:     2000                    .OPT  P,OO
40:     2000        SETLFS      =     $FFBA
50:     2000        SETNAM      =     $FFBD
60:     2000        SAVE        =     $FFD8
70:     2000                          *=    $2000
80:     2000   A9  03                 LDA   #3
90:     2002   A2  01                 LDX   #1
100:    2004   A0  01                 LDY   #1;gépi program
110:    2006'  20  BA  FF             JSR   SETLFS
120:    2009   A9  08                 LDA   #8
130:    200B   A2  24                 LDX   #<FNEV
140:    200D   A0  20                 LDY   #>FNEV
```

```
150:  200F  20  BD  FF         JSR   SETNAM
                          ;
180:  2012  A9  00             LDA   #0 ;SAVE kezdőcím
190:  2014  85  9B             STA   $9B ;elhelyezése
200:  2016  A9  38             LDA   #$38;0. lapos
210:  2018  85  9C             STA   $9C ;mutatóba
220:  201A  A9  9B             LDA   #$9B;mutató kezdőcíme
230:  201C  A2  00             LDX   #0 ;végcím alsó
240:  201E  A0  40             LDY   #$40;végcím felső
250:  2020  20  D8  FF         JSR   SAVE
280:  2023  60                 RTS
290:  2024  4B  41  52  FNEV   .ASC  "KARAKTER"
300:                           .END
```

Ha a programot fordítás után futtatjuk, akkor a szalagra tároltuk a $3800-tól kezdve áttöltött karakterkészletet. Ennek a file-nak az eredeti helyre való visszatöltése a következőképpen valósulhat meg:

LOAD"KARAKTER",1,1

Monitorból:

L "KARAKTER",1

Gépi kódú programból:

```
10:SYS  12224
30:     2000                           .OPT  P,OO
40:     2000               SETLFS      =     $FFBA
50:     2000               SETNAM      =     $FFBD
60:     2000               LOAD        =     $FFD5
70:     2000                           *=    $2000
80:     2000  A9  03                   LDA   #3
90:     2002  A2  01                   LDX   #1
100:    2004  A0  01                   LDY   #1;gépi pr. LOAD
110:    2006  20  BA  FF               JSR   SETLFS
120:    2009  A9  08                   LDA   #8
130:    200B  A2  18                   LDX   #<FNEV
140:    200D  A0  20                   LDY   #>FNEV
150:    200F  20  BD  FF               JSR   SETNAM
                               ;
180:    2012  A9  00                   LDA   #0;LOAD (1=VERIFY)
250:    2014  20  D5  FF               JSR   LOAD
280:    2017  60                       RTS
290:    2018  4B  41  52   FNEV        .ASC  "KARAKTER"
300:                                   .END
```

A szalagos egység után nézzük most meg, hogy hogyan lehet gépi programmal nyomtatni.

```
10  OPEN 5,4
20  PRINT#5,"NYOMTATAS"
30  CLOSE5
```

Gépi kódú programmal:

```
10:SYS  12224
20:     2000                            .OPT    P,OO
30:     2000                SETLFS      =       $FFBA
40:     2000                SETNAM      =       $FFBD
50:     2000                OPEN        =       $FFC0
60:     2000                CHKOUT      =       $FFC9
70:     2000                CHROUT      =       $FFD2
80:     2000                CLRCHN      =       $FFCC
90:     2000                CLOSE       =       $FFC3
100:    2000                            *=      $2000
110:    2000  A9 05                     LDA     #5
120:    2002  A2 04                     LDX     #4
130:    2004  A0 00                     LDY     #0
140:    2006  20 BA FF                  JSR     SETLFS
150:    2009  A9 00                     LDA     #0 ;nincs név
160:    200B  20 BD FF                  JSR     SETNAM
170:    200E  20 C0 FF                  JSR     OPEN
                                        ;
190:    2011  A2 05                     LDX     #5
200:    2013  20 C9 FF                  JSR     CHKOUT
210:    2016  A2 00                     LDX     #0 ;ciklus eleje
220:    2018  BD 2D 20      KIIR        LDA     SZOVEG,X;karakter
230:    201B  F0 07                     BEQ     VEGE ;ha 0, vége
240:    201D  20 D2 FF                  JSR     CHROUT ;szöveg
                                                kiírás
250:    2020  E8                        INX
260:    2021  4C 18 20                  JMP     KIIR
                            ;
280:    2024  20 CC FF      VEGE        JSR     CLRCHN
290:    2027  A9 05                     LDA     #5
300:    2029  20 C3 FF                  JSR     CLOSE
310:    202C  60                        RTS
320:    202D  4E 59 4F      SZOVEG      .ASC    "NYOMTATAS"
330:    2036  0D 00                     .BYTE   13,0
```

A nyomtató használatakor különösen ügyeljünk arra, hogy a kiküldött karaktersorozat csak akkor kerül nyomtatásra, ha betelik egy sor vagy újsorjel kerül elküldésre.

Ezek után nézzünk néhány példát az OPEN használatára lemezegység esetén. Első példánkban a 15-ös csatornára küldünk ki egy lemez-parancsot. Ez jelen esetben egy inicializálásra ad utasítást, de ez a parancs igényeinknek megfelelően bármi más is lehet.

```
10   OPEN 15,8,15
20   PRINT#15,"I"
30   CLOSE 15
```

Gépi kódú programból ez a következőképpen valósítható meg:

```
10   SYS12224
20   .OPT P,OO
30   SETLFS  =$FFBA
40   SETNAM  =$FFBD
50   OPEN    =$FFC0
60   CHKOUT  =$FFC9
70   CHROUT  =$FFD2
80   CLRCHN  =$FFCC
90   CLOSE   =$FFC3
100   *=$2000         ;indítás:SYS 8192
110   LDA #15         ;logikai file-szám
120   LDX #8          ;egységszám
130   LDY #15         ;csatornaszám
140   JSR SETLFS
150   LDA #0          ;nincs file-név
160   JSR SETNAM
170   JSR OPEN
180   ;
190   LDX #15         ;parancscsatorna
200   JSR CHKOUT      ;outputra
210   LDA #"I"        ;parancs
220   JSR CHROUT;     kiküldése
230   ;
240   JSR CLRCHN
250   LDA #15         ;15-ös csatorna
260   JSR CLOSE       ;lezárása
270   RTS
280   .END
```

A következő példaprogram már egy összetettebb feladatot valósít meg. Tetszőleges nevű és hosszúságú szekvenciális file-t listázhatunk vele a képernyőre. A file nevét input adatként kell megadnunk.

```
10   INPUT"FILE NEVE:";A$
20   OPEN 2,8,2,A$+",S,R"
30   GET#2,A$
40   PRINTA$;
```

```
50  IF ST=0 THEN 30
60  CLOSE2
```

Gépi kódú megfelelője:

```
10   SYS12224
20   9.OPT P,OO
30   SETLFS       =$FFBA
40   SETNAM       =$FFBD
50   OPEN         =$FFC0
60   CHKIN        =$FFC6
70   CHRIN        =$FFCF
80   CLRCHN       =$FFCC
90   CLOSE        =$FFC3
100  READST       =$FFB7
110  PRINT        =$FF4C
115  PRIMM        =$FF4F
120  *= $2000                ;indítás: SYS 8192
121  ;
122  JSR PRIMM               ;a soronkövetkező üzenet kiírása
124  .BYTE 13:               .ASC "FILE NEVE:":.BYTE 0
130  LDY #0
140  KEZD JSR CHRIN         ;input a képernyőről
150  CMP #13                 ;RETURN a végén ?
160  BEQ NVEG                ;igen, név vége
170  STA FNEV,Y              ;név összeállítása FNEV-től
180  INY
190  JMP KEZD                ;név következő karaktere
200  NVEG JSR PRINT          ;listázás előtt új sor jel kiírása
202  LDX #0
210  CIK LDA NEVV,X          ;név végére ",S,R"
220  STA FNEV,Y              ;átmásolása
230  INX:INY
240  CPX #4
250  BNE CIK                 ;másoló ciklus vége
260  DEY:STY NHOSSZ          ;parancs füzér hossza NHOSSZ-ba
270  LDA #2
200  LDX #8
290  LDY #2
300  JSR SETLFS
310  LDA NHOSSZ
320  LDX #<FNEV
330  LDY #>FNEV
340  JSR SETNAM
350  JSR OPEN
360  ;
```

```
370  LDX #2                             ;2-es logikai file
380  JSR CHKIN                          ;input csatornaként
390  OLV JSR CHRIN                      ;egy byte olvasása
400  JSR PRINT                          ;és kiírása képernyőre
410  JSR READST                         ;file vége ?
420  BEQ OLV                            ;nem, olvasás tovább
430  ;
440  JSR CLRCHN
450  LDA #2                             ;2-es logikai file
460  JSR CLOSE                          ;lezárása
470  RTS
480  FNEV .ASC "0000000000000000000000"
490  NEVV .ASC ",S,R"
500  NHOSSZ .BYTE 0
510  .END
```

A program elején alkalmaztuk a kiegészítő KERNAL rutinok között szereplő PRIMM rutint, mellyel kényelmesen megoldható a "FILE NEVE:" üzenet kiírása. A file-név lekérdezésekor kihasználtuk a CHRIN speciális tulajdonságát képernyő-input esetén. Az Y-regiszterben számoltuk a beírt file-név hosszát, a ",S,R" hozzáfűzése után megkapott értékét NHOSSZ-ban (mint NévHOSSZ) tároltuk. A szokásos megnyitási procedúra után addig olvastunk a file-ból, míg ST 0 volt. A képernyőre írást is a kiegészítő rutinok között szereplő PRINT rutinnal végeztük, ami kizárólag a képernyőt kezeli. A file végén a lezárás a már megszokott módon történik.

2.11.4. AZ IEC BUSZHOZ KAPCSOLÓDÓ ESZKÖZÖK HASZNÁLATA

A soros buszhoz kapcsolódó perifériák (nyomtató, lemezegység) elérése nem csak logikai file-okon keresztül lehetséges. Az eddig leírt példákban a perifériák elérését a BASIC logikáját követő OPEN utasítás használatával valósítottuk meg. A KERNAL egy másik programozási módszer használatát is lehetővé teszi, mégpedig az IEC buszon történő közvetlen adatforgalmazást.

Bekapcsolás után a C 16-os vagy a Plus/4-es perifériái az IEC buszon mind "figyelő" állapotban vannak. Ez azt jelenti, hogy semmilyen adatforgalmat sem végeznek, csupán figyelik a központi egységet, hogy számukra küld-e valamilyen parancsot. Amint valamelyik periféria megkapta a neki címzett parancsot a számítógéptől, ettől függően felkészül az adatforgalomra. Ha ez a parancs adatküldésre szólítja fel, akkor az eszköz "beszélő" állapotba kerül és felkészül az adatok küldésére. Ha a kapott parancs adatok fogadására utasít, akkor az eszköz "hallgató" állapotba kerül és adatfogadásra készül fel. Az adatforgalom befejezését jelző parancs megérkezése után a perifériák befejezik az adatküldést vagy adatfogadást és ismét "figyelő" állapotba kerülnek.

Az IEC buszon "beszélő" állapotban egyszerre csak egy egység lehet, különben az adatok zavarnák egymást. Ugyanakkor azonban több, "hallgató" állapotban lévő eszköz fogadhatja az egyetlen "beszélő" egységről érkező adatokat.

A perifériákat "hallgató" vagy "beszélő" állapotba hozó KERNAL rutinok a következők:

HALLGATÓ	BESZÉLŐ
LISTEN	TALK
SECOND	TKSA

Ezeknek a rutinoknak a használatával helyettesíthető az OPEN-nal történő file-megnyitás és a logikai file-okon történő adatforgalom.

A soros buszon történő közvetlen adatforgalmazást a következő két KERNAI rutin végzi:

ADATKÜLDÉS	ADATFOGADÁS
CIOUT	ACPTR

Az adatforgalom befejezése:

ADATKÜLDÉS	ADATFOGADÁS
UNLSN	UNTLK

Ha ilymódon szeretnénk adatokat küldeni valamelyik perifériára, akkor ennek a menete:

- a LISTEN és SECOND rutinokkal "hallgató" állapotba kell hozni az eszközt;
- az adatokat a CIOUT rutin meghívásával el kell küldeni;
- az adatküldés befejezésekor az UNLSN rutint kell meghívni;

Adatok fogadásakor értelemszerűen a TALK, TKSA, ACPTR, UNTLK rutinokat kell alkalmazni.

Első két példánkban az az egyszerűbb eset szerepel, amikor a periféria kezeléséhez file-névre nincs szükség. Például küldjünk ki 100 db "A" betűt a nyomtatóra:

```
10  OPEN5,4
20  FOR I=1 TO 100
30  PRINT#5,"A";
50  PRINT#5,CHR$(13);
60  CLOSE 5
```

Gépi kódú megfelelője:

```
100  SYS12224
110  .OPT P,OO
120  LISTEN =$FFB1
130  SECOND=$FF93
140  CIOUT  =$FFA8
150  UNLSN  =$FFAE
160  ST     =$90
170  *= $2000
180  LDA    #0          ;0 érték
190  STA    ST          ;ST változóba
200  LDA    #4          ;nyomtató egységszám
210  JSR    LISTEN      ;hallgató állapotba
220  LDA    #$60        ;másodlagos cím=0+$60
230  JSR    SECOND      ;elküldése
240  LDY    #100        ;100 db
250  LDA    #"A"        ;"A" betű
260  KIIR   JSR CIOUT   ;elküldése
270  DEY
280  BNE    KIIR        ;ciklus vége
290  LDA    #13         ;újsorjel
300  JSR    CIOUT       ;elküldése
310  JSR    UNLSN       ;adatküldés befejezése
320  RTS
330  .END
```

Abban az esetben, ha a perifériakezelésnél nem használunk file-nevet, a SECOND és TKSA rutinnal elküldött parancs byte felső négy bitje kötelezően 0110. Az alsó négy bitben van a másodlagos cím értéke. Ha például kisbetűs üzemmódban szeretnénk működtetni a nyomtatót, akkor a SECOND meghívása előtt az akkumulátorba $67 értéket kell tölteni, ami a 7-es másodlagos címnek felel meg.

A következő példaprogram a lemezegység hibacsatornáját olvassa be, és tartalmát a képernyőre írja.

```
10  OPEN 15,8,15
20  INPUT#15,A,A$,B,C
30  PRINT A;A$;B;C
40  CLOSE15
```

Egyszerűbb megoldás: PRINT DS$

Gépi kódú változat:

```
100  SYS12224
110  .OPT P,OO
120  TALK   =$FFB4
130  TKSA   =$FF96
140  ACPTR  =$FFA5
150  UNTLK  =$FFAB
160  PRINT  =$FF4C
170  READST=$FFB7
180  ST     =$90
190  *=$2000
200  LDA    #0
210  STA    ST
220  LDA    #13          ;újsorjel
230  JSR    PRINT        ;a képernyőre
240  LDA    #8           ;lemezegység
250  JSR    TALK         ;"beszélő"
260  LDA    #$6F         ;másodlagos cím
270  JSR    TKSA         ;elküldése
280  OLV    JSR ACPTR    ;egy byte olvasása
290  JSR    PRINT        ;és kiírása képernyőre
300  JSR    READST       ;file vége ?
310  BEQ    OLV          ;ha nem, újra olvasás
320  LDA    #13          ;ha igen, egy újsorjel
330  JSR    PRINT        ;a képernyőre
340  JSR    UNTLK        ;adatfogadás befejezése
350  RTS
360  .END
```

Az előző példaprogramok mindegyike olyan műveletet végzett, melyekhez nem volt szükség file-név megadásra. A következő részben a file-név (parancs füzér) elküldésére írunk néhány példát. Az első példaprogram az előzőekből már megismert szekvenciális file listázó program azzal a különbséggel, hogy a file-t nem OPEN-nal nyitjuk meg, hanem a most megismert módszerrel.

```
100  SYS12224
110  .OPT P,OO
120  LISTEN =$FFB1
130  SECOND=$FF93
140  CIOUT  =$FFA8
150  UNLSN  =$FFAE
160  TALK   =$FFB4
170  TKSA   =$FF96
180  ACPTR  =$FFA5
190  CHRIN  =$FFCF
```

```
200  ST      =$90
210  READST=$FFB7
220  PRINT   =$FF4C
230  PRIMM   =$FF4F
240  *=$2000
250  ;
260  JSR         PRIMM
270  .BYTE 13:.ASC "FILE NEVE:":.BYTE 0
280  LDY         #0
290  KEZD JSR CHRIN
300  CMP         #13           ;név vége ?
310  BEQ         NVEG          ;igen
320  STA         FNEV,Y        ;név következő karakter
330  INY
340  JMP         KEZD          ;újabb karakter
350  NVEG        JSR PRINT     ;új sor a képernyőre
360  LDX         #0
370  CIK         LDA NEVV,X    ;név + ",S,R" összeállítás
380  STA         FNEV,Y
390  INX:        INY
400  CPX         #4
410  BNE         CIK
420  DEY:        STY NHOSSZ    ;név hossza
430  ;
440  LDA         #0            ;ST
450  STA         ST            ;nullázás
460  LDA         #8            ;egységszám,
470  JSR         LISTEN
480  LDA         #$F8          ;másodlagos cím IEC OPEN-hoz
490  JSR         SECOND
500  LDX         #0
510  NEVKI       LDA FNEV,X    ;név elküldése
520  JSR         CIOUT
530  CPX         NHOSSZ
540  BEQ         NEVVEG
550  INX:        JMP NEVKI
560  NEVVEG      JSR UNLSN     ;név elküldés vége
570  ;
580  LDA         #8
590  JSR         TALK
600  LDA         #$68
610  JSR         TKSA
620  OLV         JSR ACPTR     ;olvasás a file-ból
630  JSR         PRINT
640  JSR         READST        ;file vége ?
```

```
650  BEQ     OLV           ;nem, olvasás
660  ;
670  LDA     #8            ;IEC CLOSE: egység
680  JSR     LISTEN
690  LDA     #$E8          ;másodlagos cím + IEC CLOSE
700  JSR     SECOND        ;elküldés
710  JSR     UNLSN
720  RTS
730  FNEV    .ASC "0000000000000000000000"
740  NEVV    .ASC ",S,R"
750  NHOSSZ  .BYTE 0
760  .END
```

A paraméter füzér (file-név) elküldésének menete a következő:

1) LISTEN elküldése, AC = egységszám.

2) SECOND elküldése, AC felső négy bitje 1111 (= OPEN).

3) File-név elküldése CIOUT-tal.

4) UNLSN elküldése.

Ezután a csatornát céljainknak megfelelően "beszélő" vagy "hallgató" állapotba hozhatjuk, és a megszokott módon küldhetünk vagy fogadhatunk adatokat.

Az ilyen (file-névvel megnyitott) csatornát az adatforgalom után nem elég a megszokott módon lezárni. Ennek menete a következő:

1) LISTEN elküldése, AC = egységszám.

2) SECOND elküldése, AC felső négy bitje 1110 (= CLOSE).

3) UNLSN elküldése.

Utolsó példánkban az eddig megismert kétféle perifériakezelési módszer párhuzamos használatát mutatjuk be. A program a lemezegység tartalomjegyzékének első blokkját olvassa be és a képernyőmemóriába írja. Az adatforgalom az OPEN-nel megnyitott 2-es csatornán folyik, míg a DOS-parancsokat a 15-ös csatornára közvetlenül, OPEN nélkül küldjük ki.

A kétféle módszer egy csatornán belül is kombinálható: a 2-es csatornát OPEN-nel nyitottuk meg, és az adatokat az ACPTR rutinnal olvastuk be.

```
100  SYS12224
110  .OPT P,OO
120  LISTEN  =$FFB1
130  SECOND  =$FF93
140  CIOUT   =$FFA8
150  UNLSN   =$FFAE
160  TALK    =$FFB4
170  TKSA    =$FF96
180  ACPTR   =$FFA5
```

```
190  CHRIN  =$FFCF
200  SETLFS =$FFBA
210  OPEN   =$FFC0
220  PRINT  =$FF4C
230  SETNAM=$FFBD
240  CLOSE  =$FFC3
250  CLRCHN=$FFCC
260  *=$2000
270  ;
280  LDA      #2
290  LDX      #8
300  LDY      #2
310  JSR      SETLFS
320  LDA      #1
330  LDX      #<NEV
340  LDY      #>NEV
350  JSR      SETNAM
360  JSR      OPEN         ;OPEN2,8,2,"#"
370  ;
380  LDA      #8
390  JSR      LISTEN
400  LDA      #$6F
410  JSR      SECOND
420  LDY      #0
430  U1K      LDA BR,Y
440  JSR      CIOUT        ;PRINT#15,"U1:2 0 18 1"
450  INY
460  CPY      #11
470  BNE      U1K
480  LDA      #8
490  JSR      UNLSN
500  ;
510  LDA      #8
520  JSR      LISTEN
530  LDA      #$6F
540  JSR      SECOND
550  LDY      #0
560  BPK      LDA BP,Y
570  JSR      CIOUT        ;PRINT#15,"B-P:2 0"
580  INY
590  CPY      #7
600  BNE      BPK
610  LDA      #8
620  JSR      UNLSN
625
```

```
630  LDA    #8
640  JSR    TALK
650  LDA    #$62
660  JSR    TKSA
670  JSR    $DC8C          ;új sor a képernyőre
680  LDY    #0
690  BEOL   JSR ACPTR      ;1 byte beolvasása
700  STA    $0C00,Y        ;kiírása
710  INY
720  BNE    BEOL
730  JSR    CLRCHN
740  LDA    #2
750  JSR    CLOSE
760  RTS
770  NEV    .ASC "#"
780  BR .ASC "U1:2 0 18 1"
790  BP  .ASC "B-P:2 0"
800  .END
```

A leggyakrabban használt parancsbyte-ok

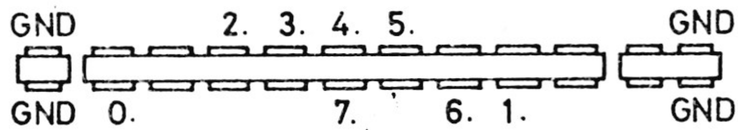
Felső 4 bit	Alsó 4 bit	Jelentés
0010	egységszám	LISTEN
0011	1111	UNLSN
0100	egységszám	TALK
0101	1111	UNTLK
0110	másodl. cím	SECOND
0110	másodl. cím	TKSA
1110	csatornaszám	CLOSE
1111	0 = LOAD 1 = SAVE 2–14 = adatcsat. 15 = parancs-csat.	OPEN

2.11.5. INPUT-OUTPUT CÍMEK

0000	Processzor adatirányregisztere
0001	Processzor I/O kapu Az egyes bitek jelentése: 0. soros busz DATA OUT 1. soros busz CLK OUT szalag írása 2. soros busz ATN OUT 3. szalagos egység motor vezérlése 4. szalag olvasás 5. – 6. soros busz CLK IN 7. soros busz DATA IN
FD00–FD0F	C 16-os: nem használt Plus/4-es: RS232 regiszterek FD00 adat FD01 állapotregiszter FD02 parancsregiszter FD03 kontrollregiszter FD04–FD0F az előbbi 4 regiszter ismétlődik
FD10–FD1F	Szalagos egység nyomógomb-lekérdezése (16-szor ismétlődik) FF=nincs lenyomva FB=lenyomva valamelyik Csak Plus/4-esen: USER PORT-regiszter 8-vonalú, kétirányú kapcsolódási lehetőséget ad a felhasználó számára (például egyszerűbb vezérlési feladatra)
FD20–FD2F	Nem használt
FD30–FD3F	Billentyűzet mátrix sor kiválasztása (16-szor ismétlődik) Részletes leírása a billentyűzetről szóló alfejezetben.
FD40–FDCF	Nem használt
FDD0–FDDF	ROM lapozási címek Írásukkal a belső és a külső ROM-ok közül választhatunk. Részletes leírása a memórialapozásról szóló alfejezetben.
FDE0–FEBF	Nem használt
FEC0–FEFF	Párhuzamos IEC terület Használt címek: FEC0–FEC5 1. egység

FED0–FED5 2. egység
FEE0–FEE5 3. egység
FEF0–FEF5
FF00–FF3F TED-chip regiszterei

A $FD10 bitjei és a felhasználói csatlakozó (USER PORT) kivezetései közötti kapcsolat:



2.12. A TED chip regisztertérképe

FF00 1. időzítő alsó byte :beírt értékről indul újra
FF01 1. időzítő felső byte

FF02 2. időzítő alsó byte :FF-ről indul újra
FF03 2. időzítő felső byte

FF04 3. időzítő alsó byte :FF-ről indul újra
FF05 3. időzítő felső byte

FF06 TED vezérlő regiszter:
7. I.C teszt: értéke 0
6. 1 = bővített háttérszín mód
5. 1 = bittérkép mód
4. 1 = képernyő engedélyezve
3. 1 = 25 soros képernyő
0 = 24 soros képernyő
2.–0. függőleges finom scroll

FF07 TED vezérlő regiszter
7. 0 = inverz megjelenítés
1 = inverz megjelenítés letiltva: 256 karakter
6. 0 = PAL (európai) szabvány TV
1 = NTSC (amerikai) szabvány TV
5. 1 = TED STOP
4. 1 = többszín üzemmód
3. 1 = 40 karakter széles képernyő
0 = 38 karakter széles képernyő
2.–0. vízszintes finom scroll

FF08 Billentyűzet állapot (latch)
FF09 Megszakítás érzékelő
7. Megszakítás történt
6. 3. időzítő megszakítása
5. -
4. 2. időzítő megszakítása
3. 1. időzítő megszakítása
2. fénytoll megszakítása
1. raszterregiszter megszakítása
0. -

FF0A Megszakítás engedélyező: 1 = engedélyezve
7. -
6. 3. időzítő megszakítás engedélyezés
5. -
4. 2. időzítő megszakítás engedélyezés
3. 1. időzítő megszakítás engedélyezés
2. fénytoll megszakítás engedélyezve
1. raszterregiszter megszakítás engedélyezve
0. raszterregiszter 8. bitje

FF0B Rasztermegszakítás helye (legfelső bit az FF0A-ban van)

FF0C Hardver kurzorpozíció felső rész
7.–2. –
1.–0. kurzorpozíció 9. és 8. bitje

FF0D Hardver kurzorpozíció alsó rész
0 = bal felső sarok
03E7-től fölfelé nem látható

FF0E 1. hanggenerátor frekvencia alsó byte-ja

FF0F 2. hanggenerátor frekvencia alsó byte-ja

FF10 2. hanggenerátor frekvencia felső része
7.–2. –
1.–0. 2. hanggenerátor frekvencia felső része

FF11 Hang vezérlő regiszter
7. 0 = négyszögjel kimenet
1 = analóg, a hangerővel szabályozott jelalak
6. 1 = 2. hanggenerátor zaj engedélyezett
5. 1 = 2. hanggenerátor hang engedélyezett
4. 1 = 1. hanggenerátor hang engedélyezett
3.–0. hangerő: 0-tól 8-ig növelhető

FF12 7. –

6. –
5.–3. bittérkép helye 8K-ban
2. 0 = karaktergenerátor, bittérkép RAM-ban
1 = ROM-ban van
1.–0. 1. hanggenerátor felső 2 bit

FF13 7.–2. karaktergenerátor kezdőcíme K-ban
1. single CLOCK
0. RAM–ROM lapozás állapota olvasható ki belőle:
0 = RAM 1 = ROM

FF14 Képernyőmemória kezdete
7.–3. Szín- és képernyőmemória vagy
fényerő- és színmemória kezdete 2K-ban
2.–0. –

FF15 Háttérszínregiszter
7. –
6.–4. fényerő
3.–0. szín

FF16 Segédszínregiszter 1.
7. –
6.–4. fényerő
3.–0. szín

FF17 Segédszínregiszter 2.
7. –
6.–4. fényerő
3.–0. szín

FF18 Segédszínregiszter 3.
7. –
6.–4. fényerő
3.–0. szín

FF19 Keretszínregiszter
7. –
6.–4. fényerő
3.–0. szín

FF1A Karakterpozíció kiírás regiszter felső része
7.–2. –
1.–0. karakterpozíció kiírás felső része

FF1B Karakterpozíció kiírás regiszter alsó része
értéke kiírt karaktersoronként 40-nel nő

FF1C	Videosor számláló regiszter felső része 7.–1. – 0. videosor számláló 8. bit
FF1D	Videosor számláló regiszter alsó része
FF1E	Vízszintes videopozíció számláló felső 8 bit
FF1F	7. – 6.–3. villogás időzítésszámláló (sebesség: 2 Hz) 2.–0. karaktersoron belüli rasztersorszámláló
FF20-FF3D	Nincs regiszter
FF3E	ROM kiválasztás Erre a címre bármit írva a ROM felülre lapozódik
FF3F	RAM kiválasztás Erre a címre bármit írva a RAM felülre lapozódik

Nem használható RAM: 0000–0001
FD00–FF3F

FÜGGELÉK

1. Képernyőmemória

0C00
0C28
0C50
0C78
0CA0
0CC8
0CF0
0D18
0D40
0D68
0D90
0DB8
0DE0
0E08
0E30
0E58
0E80
0EA8
0ED0
0EF8
0F20
0F48
0F70
0F98
0FC0

2. Szín- és fényerőmemória

0800
0828
0850
0878
08A0
08C8
08F0
0918
0940
0968
0990
09B8
09E0
0A08
0A30
0A58
0A80
0AA8
0AD0
0AF8
0B20
0B48
0B70
0B98
0BC0

3. Képernyő kódok

HEX	DEC	KISB.	GRAF.
00	0	@	@
01	1	a	A
02	2	b	B
03	3	c	C
04	4	d	D
05	5	e	E
06	6	f	F
07	7	g	G
08	8	h	H
09	9	i	I
0A	10	j	J
0B	11	k	K
0C	12	l	L
0D	13	m	M
0E	14	n	N
0F	15	o	O
10	16	p	P
11	17	q	Q
12	18	r	R
13	19	s	S
14	20	t	T
15	21	u	U
16	22	v	V
17	23	w	W
18	24	x	X
19	25	y	Y
1A	26	z	Z
1B	27	[	[
1C	28	£	£
1D	29	]	]
1E	30	↑	↑
1F	31	←	←
20	32		
21	33	!	!
22	34	"	"
23	35	#	#
24	36	$	$
25	37	%	%
26	38	&	&
27	39	'	'
28	40	(	(
29	41	)	)
2A	42	*	*
2B	43	+	+
2C	44	,	,
2D	45	-	-
2E	46	.	.
2F	47	/	/
30	48	0	0
31	49	1	1
32	50	2	2
33	51	3	3
34	52	4	4
35	53	5	5
36	54	6	6
37	55	7	7
38	56	8	8
39	57	9	9
3A	58	:	:
3B	59	;	;
3C	60	<	<
3D	61	=	=
3E	62	>	>
3F	63	?	?

HEX	DEC	KISB.	GRAF.
40	64		
41	65	A	
42	66	B	
43	67	C	
44	68	D	
45	69	E	
46	70	F	
47	71	G	
48	72	H	
49	73	I	
4A	74	J	
4B	75	K	
4C	76	L	
4D	77	M	
4E	78	N	
4F	79	O	
50	80	P	
51	81	Q	
52	82	R	
53	83	S	
54	84	T	
55	85	U	
56	86	V	
57	87	W	
58	88	X	
59	89	Y	
5A	90	Z	
5B	91		
5C	92		
5D	93		
5E	94		
5F	95		
60	96		
61	97		
62	98		
63	99		
64	100		
65	101		
66	102		
67	103		
68	104		
69	105		
6A	106		
6B	107		
6C	108		
6D	109		
6E	110		
6F	111		
70	112		
71	113		
72	114		
73	115		
74	116		
75	117		
76	118		
77	119		
78	120		
79	121		
7A	122		
7B	123		
7C	124		
7D	125		
7E	126		
7F	127		

4. ASCII kódtáblázat

4.1. ASCII VEZÉRLŐKÓDOK

HEX	'C	BILL.		HATÁS
00	0	–		–
01	1	<CT>	A	–
02	2	<CT>	B	–
03	3	STOP		BASIC PROGRRAM MEGSZAKÍTÁSA
		<CT>	C	–
04	4	<CT>	D	–
05	5	<CT>	2	FEHÉR ($71)
		<CT>	E	
06	6	<CT>	F	–
		<CT>	=	
07	7	<CT>	G	–
08	8	<CT>	H	<SH><C=> LETILTÁS
09	9	<CT>	I	<SH><C=> ENGEDÉLYEZÉS
0A	10	<CT>	J	–
0B	11	<CT>	K	–
0C	12	<CT>	L	–
0D	13	RETURN		ÚJ SOR ELEJE
		<CT>	M	
0E	14	<CT>	N	KISBETŰ/NAGYBETŰ MÓD
0F	15	<CT>	O	–
10	16	<CT>	P	–
11	17	CRSR	LE	KURZOR LE
		<CT>	Q	
12	18	<CT>	9	RVS ON
		<CT>	R	
13	19	HOME		KURZOR A BAL FELSŐ SAROKBA
14	20	DEL		TÖRLÉS BALRA
		<CT>	T	
15	21	<CT>	U	–
16	22	<CT>	V	–
17	23	<CT>	W	–
18	24	<CT>	X	–
19	25	<CT>	Y	–
1A	26	<CT>	Z	–
1B	27	ESC		ESCAPE MÓD
		<CT>		

HEX	DEC	BILL.		HATÁS
1C	28	<CT>	3	PIROS ($32)
		<CT>	—	
1D	29	CRSR	JOBBRA	KURZOR JOBBRA
		<CT>	;	
1E	30	<CT>	6	ZÖLD ($35)
1F	31	<CT>	7	KÉK ($46)
80	128			–
81	129	<C=>	1	NARANCS ($48)
82	130	<CT>	,	FLASH ON
83	131	<SH>	STOP	dL"* + RUN
84	132	<CT>		FLASH OFF
85	133	F1		
86	134	F3		
87	135	<SH>	F2	
88	136	<SH>	HELP	
89	137	F2		
8A	138	<SH>	F1	
8B	139	<SH>	F3	
8C	140	HELP		
8D	141	<SH>	RET	
8E	142	–		NAGYBETŰ/GRAFIKUS MÓD
8F	143	–		
90	144	<CT>	1	FEKETE (00)
91	145	CRSR	FEL	KURZOR FEL
92	146	<CT>	0	RVS OFF
93	147	<SH>	CLEAR	KÉPERNYŐ TÖRLÉS
94	148	<SH>	INST	BESZÚRÁS
95	149	<C=>	2	BARNA ($29)
96	150	<C=>	3	SÁRGÁSZÖLD ($5A)
97	151	<C=>	4	RÓZSASZÍN ($6B)
98	152	<C=>	5	KÉKESZÖLD ($5C)
99	153	<C=>	6	VILÁGOSKÉK ($6D)
9A	154	<C=>	7	SÖTÉTKÉK ($2E)
9B	155	<C=>	8	VILÁGOS ZÖLD ($5F)
9C	156	<CT>	5	BÍBOR ($44)
9D	157	CRSR	BAL	KURZOR BALRA
9E	158	<CT>	8	SÁRGA ($77)
9F	159	<CT>	4	TÜRKIZ ($63)

ASCII KÓDOK

HEX	DEC	KISB.	GRAF.
20	32		
21	33	!	!
22	34	"	"
23	35	#	#
24	36	$	$
25	37	%	%
26	38	&	&
27	39	'	'
28	40	(	(
29	41	)	)
2A	42	*	*
2B	43	+	+
2C	44	,	,
2D	45	-	-
2E	46	.	.
2F	47	/	/
30	48	0	0
31	49	1	1
32	50	2	2
33	51	3	3
34	52	4	4
35	53	5	5
36	54	6	6
37	55	7	7
38	56	8	8
39	57	9	9
3A	58	:	:
3B	59	;	;
3C	60	<	<
3D	61	=	=
3E	62	>	>
3F	63	?	?

HEX	DEC	KISB.	GRAF.
40	64	@	@
41	65	a	A
42	66	b	B
43	67	c	C
44	68	d	D
45	69	e	E
46	70	f	F
47	71	g	G
48	72	h	H
49	73	i	I
4A	74	j	J
4B	75	k	K
4C	76	l	L
4D	77	m	M
4E	78	n	N
4F	79	o	O
50	80	p	P
51	81	q	Q
52	82	r	R
53	83	s	S
54	84	t	T
55	85	u	U
56	86	v	V
57	87	w	W
58	88	x	X
59	89	y	Y
5A	90	z	Z
5B	91	[	[
5C	92	£	£
5D	93	]	]
5E	94	↑	↑
5F	95	←	←

HEX	DEC	KISB.	GRAF.
60	96		
61	97	A	
62	98	B	
63	99	C	
64	100	D	
65	101	E	
66	102	F	
67	103	G	
68	104	H	
69	105	I	
6A	106	J	
6B	107	K	
6C	108	L	
6D	109	M	
6E	110	N	
6F	111	O	
70	112	P	
71	113	Q	
72	114	R	
73	115	S	
74	116	T	
75	117	U	
76	118	V	
77	119	W	
78	120	X	
79	121	Y	
7A	122	Z	
7B	123		
7C	124		
7D	125		
7E	126		
7F	127		

HEX	DEC	KISB.	GRAF.
A0	160		
A1	161		
A2	162		
A3	163		
A4	164		
A5	165		
A6	166		
A7	167		
A8	168		
A9	169		
AA	170		
AB	171		
AC	172		
AD	173		
AE	174		
AF	175		
B0	176		
B1	177		
B2	178		
B3	179		
B4	180		
B5	181		
B6	182		
B7	183		
B8	184		
B9	185		
BA	186		
BB	187		
BC	188		
BD	189		
BE	190		
BF	191		

HEX	DEC	KISB.	GRAF.
C0	192		
C1	193	A	
C2	194	B	
C3	195	C	
C4	196	D	
C5	197	E	
C6	198	F	
C7	199	G	
C8	200	H	
C9	201	I	
CA	202	J	
CB	203	K	
CC	204	L	
CD	205	M	
CE	206	N	
CF	207	O	
D0	208	P	
D1	209	Q	
D2	210	R	
D3	211	S	
D4	212	T	
D5	213	U	
D6	214	V	
D7	215	W	
D8	216	X	
D9	217	Y	
DA	218	Z	
DB	219		
DC	220		
DD	221		
DE	222		π
DF	223		
E0	224		
E1	225		
E2	226		
E3	227		
E4	228		
E5	229		
E6	230		
E7	231		
E8	232		
E9	233		
EA	234		
EB	235		
EC	236		
ED	237		
EE	238		
EF	239		
F0	240		
F1	241		
F2	242		
F3	243		
F4	244		
F5	245		
F6	246		
F7	247		
F8	248		
F9	249		
FA	250		
FB	251		
FC	252		
FD	253		
FE	254		
FF	255		π

5. BASIC kulcsszavak és tokenjeik

HEX	DEC	NÉV
$80	128	END
$81	129	FOR
$82	130	NEXT
$83	131	DATA
$84	132	INPUT#
$85	133	INPUT
$86	134	DIM
$87	135	READ
$88	136	LET
$89	137	GOTO
$8A	138	RUN
$8B	139	IF
$8C	140	RESTORE
$8D	141	GOSUB
$8E	142	RETURN
$8F	143	REM
$90	144	STOP
$91	145	ON
$92	146	WAIT
$93	147	LOAD
$94	148	SAVE
$95	149	VERIFY
$96	150	DEF
$97	151	POKE
$98	152	PRINT#
$99	153	PRINT
$9A	154	CONT
$9B	155	LIST
$9C	156	CLR
$9D	157	CMD
$9E	158	SYS
$9F	159	OPEN
$A0	160	CLOSE
$A1	161	GET
$A2	162	NEW
$A3	163	TAB(
$A4	164	TO
$A5	165	FN
$A6	166	SPC(
$A7	167	THEN

HEX	DEC	NÉV
$A8	168	NOT
$A9	169	STEP
$AA	170	+
$AB	171	–
$AC	172	*
$AD	173	/
$AE	174	↑
$AF	175	AND
$B0	176	OR
$B1	177	>
$B2	178	=
$B3	179	<
$B4	180	SGN
$B5	181	INT
$B6	182	ABS
$B7	183	USR
$B8	184	FRE
$B9	185	POS
$BA	186	SQR
$BB	187	RND
$BC	188	LOG
$BD	189	EXP
$BE	190	COS
$BF	191	SIN
$C0	192	TAN
$C1	193	ATN
$C2	194	PEEK
$C3	195	LEN
$C4	196	STR$
$C5	197	VAL
$C6	198	ASC
$C7	199	CHR$
$C8	200	LEFT$
$C9	201	RIGHT$
$CA	202	MID$
$CB	203	GO
$CC	204	RGR
$CD	205	RCLR
$CE	206	RLUM
$CF	207	JOY
$D0	208	RDOT
$D1	209	DEC
$D2	210	HEX$

HEX	DEC	NÉV
$D3	211	ERR$
$D4	212	INSTR
$D5	213	ELSE
$D6	214	RESUME
$D7	215	TRAP
$D8	216	TRON
$D9	217	TROFF
$DA	218	SOUND
$DB	219	VOL
$DC	220	AUTO
$DD	221	PUDEF
$DE	222	GRAPHIC
$DF	223	PAINT
$E0	224	CHAR
$E1	225	BOX
$E2	226	CIRCLE
$E3	227	GSHAPE
$E4	228	SSHAPE
$E5	229	DRAW
$E6	230	LOCATE
$E7	231	COLOR
$E8	232	SCNCLR
$E9	233	SCALE
$EA	234	HELP
$EB	235	DO
$EC	236	LOOP
$ED	237	EXIT
$EE	238	DIRECTORY
$EF	239	DSAVE
$F0	240	DLOAD
$F1	241	HEADER
$F2	242	SCRATCH
$F3	243	COLLECT
$F4	244	COPY
$F5	245	RENAME
$F6	246	BACKUP
$F7	247	DELETE
$F8	248	RENUMBER
$F9	249	KEY
$FA	250	MONITOR
$FB	251	USING
$FC	252	UNTIL
$FD	253	WHILE
$FE	254	
$FF	255	pi

6. Frekvenciatáblázat

HANG	FREKV.	SOUND	FELSŐ BYTE	ALSÓ BYTE
A	110	7	00	07
	117	64	00	40
H	123	118	00	76
Kis oktáv				
C	131	169	00	A9
	139	217	00	D9
D	147	262	01	06
	156	305	01	31
E	165	345	01	59
F	175	383	01	7F
	185	419	01	A3
G	196	453	01	C5
	208	485	01	E5
A	220	516	02	04
	233	544	02	20
H	247	571	02	3B
I. oktáv				
C	262	597	02	55
	277	621	02	6D
D	294	643	02	83
	311	665	02	99
E	330	685	02	AD
F	349	704	02	C0
	370	722	02	D2
G	392	739	02	E3
	415	755	02	F3
A	440	770	03	02
	466	784	03	10
H	494	798	03	1E
II. oktáv				
C	523	810	03	2A
	554	822	03	36

HANG	FREKV.	SOUND	FELSŐ BYTE	ALSÓ BYTE
D	587	834	03	42
	622	844	03	4C
E	659	854	03	56
F	698	864	03	60
	740	873	03	69
G	784	881	03	71
	831	889	03	79
A	880	897	03	81
	932	904	03	88
H	988	911	03	8F
III. oktáv				
C	1047	917	03	95
	1109	923	03	9B
D	1175	929	03	A1
	1245	934	03	A6
E	1319	939	03	AB
F	1397	944	03	B0
	1480	948	03	B4
G	1568	953	03	B9
	1661	957	03	BD
A	1760	960	03	C0
	1865	964	03	C4
H	1976	967	03	C7
IV. oktáv				
C	2093	971	03	CB
	2217	974	03	CE
D	2349	976	03	D0
	2489	979	03	D3
E	2637	982	03	D6
F	2794	984	03	D8
	2960	986	03	DA
G	3136	988	03	DC
	3322	990	03	DE
A	3520	992	03	E0
	3729	994	03	E2
H	3951	996	03	E4

HANG	FREKV.	SOUND	FELSŐ BYTE	ALSÓ BYTE
V. oktáv				
C	4186	997	03	E5
	4435	999	03	E7
D	4699	1000	03	E8
	4978	1002	03	EA
E	5274	1003	03	EB
F	5588	1004	03	EC
	5920	1005	03	ED
G	6272	1006	03	EE
	6645	1007	03	EF

7. Átváltási táblázat

DEC	HEX	BINÁRIS
0	00	00000000
1	01	00000001
2	02	00000010
3	03	00000011
4	04	00000100
5	05	00000101
6	06	00000110
7	07	00000111
8	08	00001000
9	09	00001001
10	0A	00001010
11	0B	00001011
12	0C	00001100
13	0D	00001101
14	0E	00001110
15	0F	00001111
16	10	00010000
17	11	00010001
18	12	00010010
19	13	00010011
20	14	00010100
21	15	00010101
22	16	00010110
23	17	00010111

DEC	HEX	BINÁRIS
24	18	00011000
25	19	00011001
26	1A	00011010
27	1B	00011011
28	1C	00011100
29	1D	00011101
30	1E	00011110
31	1F	00011111
32	20	00100000
33	21	00100001
34	22	00100010
35	23	00100011
36	24	00100100
37	25	00100101
38	26	00100110
39	27	00100111
40	28	00101000
41	29	00101001
42	2A	00101010
43	2B	00101011
44	2C	00101100
45	2D	00101101
46	2E	00101110
47	2F	00101111
48	30	00110000
49	31	00110001
50	32	00110010
51	33	00110011
52	34	00110100
53	35	00110101
54	36	00110110
55	37	00110111
56	38	00111000
57	39	00111001
58	3A	00111010
59	3B	00111011
60	3C	00111100
61	3D	00111101
62	3E	00111110
63	3F	00111111
64	40	01000000
65	41	01000001
66	42	01000010

DEC	HEX	BINÁRIS
67	43	01000011
68	44	01000100
69	45	01000101
70	46	01000110
71	47	01000111
72	48	01001000
73	49	01001001
74	4A	01001010
75	4B	01001011
76	4C	01001100
77	4D	01001101
78	4E	01001110
79	4F	01001111
80	50	01010000
81	51	01010001
82	52	01010010
83	53	01010011
84	54	01010100
85	55	01010101
86	56	01010110
87	57	01010111
88	58	01011000
89	59	01011001
90	5A	01011010
91	5B	01011011
92	5C	01011100
93	5D	01011101
94	5E	01011110
95	5F	01011111
96	60	01100000
97	61	01100001
98	62	01100010
99	63	01100011
100	64	01100100
101	65	01100101
102	66	01100110
103	67	01100111
104	68	01101000
105	69	01101001
106	6A	01101010
107	6B	01101011
108	6C	01101100
109	6D	01101101

DEC	HEX	BINÁRIS
110	6E	01101110
111	6F	01101111
112	70	01110000
113	71	01110001
114	72	01110010
115	73	01110011
116	74	01110100
117	75	01110101
118	76	01110110
119	77	01110111
120	78	01111000
121	79	01111001
122	7A	01111010
123	7B	01111011
124	7C	01111100
125	7D	01111101
126	7E	01111110
127	7F	01111111

			2-ES KOMPL.	
DEC	HEX	BINÁRIS	DEC	HEX
128	80	10000000	–128	–80
129	81	10000001	–127	–7F
130	82	10000010	–126	–7E
131	83	10000011	–125	–7D
132	84	10000100	–124	–7C
133	85	10000101	–123	–7B
134	86	10000110	–122	–7A
135	87	10000111	–121	–79
136	88	10001000	–120	–78
137	89	10001001	–119	–77
138	8A	10001010	–118	–76
139	8B	10001011	–117	–75
140	8C	10001100	–116	–74
141	8D	10001101	–115	–73
142	8E	10001110	–114	–72
143	8F	10001111	–113	–71
144	90	10010000	–112	–70
145	91	10010001	–111	–6F
146	92	10010010	–110	–6E

			2-ES KOMPL.	
DEC	HEX	BINÁRIS	DEC	HEX
147	93	10010011	–109	–6D
148	94	10010100	–108	–6C
149	95	10010101	–107	–6B
150	96	10010110	–106	–6A
151	97	10010111	–105	–69
152	98	10011000	–104	–68
153	99	10011001	–103	–67
154	9A	10011010	–102	–66
155	9B	10011011	–101	–65
156	9C	10011100	–100	–64
157	9D	10011101	–99	–63
158	9E	10011110	–98	–62
159	9F	10011111	–97	–61
160	A0	10100000	–96	–60
161	A1	10100001	–95	–5F
162	A2	10100010	–94	–5E
163	A3	10100011	–93	–5D
164	A4	10100100	–92	–5C
165	A5	10100101	–91	–5B
166	A6	10100110	–90	–5A
167	A7	10100111	–89	–59
168	A8	10101000	–88	–58
169	A9	10101001	–87	–57
170	AA	10101010	–86	–56
171	AB	10101011	–85	–55
172	AC	10101100	–84	–54
173	AD	10101101	–83	–53
174	AE	10101110	–82	–52
175	AF	10101111	–81	–51
176	B0	10110000	–80	–50
177	B1	10110001	–79	–4F
178	B2	10110010	–78	–4E
179	B3	10110011	–77	–4D
180	B4	10110100	–76	–4C
181	B5	10110101	–75	–4B
182	B6	10110110	–74	–4A
183	B7	10110111	–73	–49
184	B8	10111000	–72	–48
185	B9	10111001	–71	–47
186	BA	10111010	–70	–46
187	BB	10111011	–69	–45
188	BC	10111100	–68	–44

			2-ES KOMPL.	
DEC	HEX	BINÁRIS	DEC	HEX
189	BD	10111101	–67	–43
190	BE	10111110	–66	–42
191	BF	10111111	–65	–41
192	C0	11000000	–64	–40
193	C1	11000001	–63	–3F
194	C2	11000010	–62	–3E
195	C3	11000011	–61	–3D
196	C4	11000100	–60	–3C
197	C5	11000101	–59	–3B
198	C6	11000110	–58	–3A
199	C7	11000111	–57	–39
200	C8	11001000	–56	–38
201	C9	11001001	–55	–37
202	CA	11001010	–54	–36
203	CB	11001011	–53	–35
204	CC	11001100	–52	–34
205	CD	11001101	–51	–33
206	CE	11001110	–50	–32
207	CF	11001111	–49	–31
208	D0	11010000	–48	–30
209	D1	11010001	–47	–2F
210	D2	11010010	–46	–2E
211	D3	11010011	–45	–2D
212	D4	11010100	–44	–2C
213	D5	11010101	–43	–2B
214	D6	11010110	–42	–2A
215	D7	11010111	–41	–29
216	D8	11011000	–40	–28
217	D9	11011001	–39	–27
218	DA	11011010	–38	–26
219	DB	11011011	–37	–25
220	DC	11011100	–36	–24
221	DD	11011101	–35	–23
222	DE	11011110	–34	–22
223	DF	11011111	–33	–21
224	E0	11100000	–32	–20
225	E1	11100001	–31	–1F
226	E2	11100010	–30	–1E
227	E3	11100011	–29	–1D
228	E4	11100100	–28	–1C
229	E5	11100101	–27	–1B

			2-ES KOMPL.	
DEC	HEX	BINÁRIS	DEC	HEX
230	E6	11100110	–26	–1A
231	E7	11100111	–25	–19
232	E8	11101000	–24	–18
233	E9	11101001	–23	–17
234	EA	11101010	–22	–16
235	EB	11101011	–21	–15
236	EC	11101100	–20	–14
237	ED	11101101	–19	–13
238	EE	11101110	–18	–12
239	EF	11101111	–17	–11
240	F0	11110000	–16	–10
241	F1	11110001	–15	–0F
242	F2	11110010	–14	–0E
243	F3	11110011	–13	–0D
244	F4	11110100	–12	–0C
245	F5	11110101	–11	–0B
246	F6	11110110	–10	–0A
247	F7	11110111	–9	–09
248	F8	11111000	–8	–08
249	F9	11111001	–7	–07
250	FA	11111010	–6	–06
251	FB	11111011	–5	–05
252	FC	11111100	–4	04
253	FD	11111101	–3	–03
254	FE	11111110	–2	–02
255	FF	11111111	–1	–01

8. Gépi utasítások összefoglaló táblázata

ADC – Összeadás CARRY-vel

A = A + M + C NV....ZC

Utasítás	Kód	Hossz	Ciklusszám
ADC #ee	69	2	2
ADC cc	65	2	3
ADC cc,X	75	2	4
ADC cccc	6D	3	4
ADC cccc,X	7D	3	4 + 1
ADC cccc,Y	79	3	4 + 1
ADC (cc,X)	61	2	6
ADC (cc),Y	71	2	5 + 1

AND – Logikai ÉS a tár és az akkumulátor között

A = A AND M N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
AND #ee	29	2	2
AND cc	25	2	3
AND cc,X	35	2	4
AND cccc	2D	3	4
AND cccc,X	3D	3	4 + 1
AND cccc,Y	39	3	4 + 1
AND (cc,X)	21	2	6
AND (cc),Y	31	2	5

ASL – Bitenkénti léptetés balra

C ← 76543210 ← 0 N.....ZC

Utasítás	Kód	Hossz	Ciklusszám
ASL	0A	1	2
ASL cc	06	2	5
ASL cc,X	16	2	6
ASL cccc	0E	3	6
ASL cccc,X	1E	3	7

BCC – Ugrás, ha a CARRY 0

ugrás, ha C = 0			....
Utasítás	Kód	Hossz	Ciklusszám
BCC rr	90	2	2 + 2

BCS – Ugrás, ha a CARRY 1

ugrás, ha C = 1			....
Utasítás	Kód	Hossz	Ciklusszám
BCS rr	B0	2	2 + 2

BEQ – Ugrás, ha egyenlő (nulla)

ugrás, ha Z = 1			....
Utasítás	Kód	Hossz	Ciklusszám
BEQ rr	F0	2	2 + 2

BIT – A tár tesztelése az akkumulátorral

Z = A AND M;N = Mb7;V = Mb6;			NV....Z.
Utasítás	Kód	Hossz	Ciklusszám
BIT cc	24	2	3
BIT cccc	2C	3	4

BMI – Ugrás, ha negatív

ugrás, ha N = 1			....
Utasítás	Kód	Hossz	Ciklusszám
BMI rr	30	2	2 + 2

BNE – Ugrás, ha nem egyenlő (nem nulla)

ugrás, ha Z = 0			....
Utasítás	Kód	Hossz	Ciklusszám
BNE rr	D0	2	2 + 2

BPL – Ugrás, ha pozitív

ugrás, ha N = 0			....
Utasítás	Kód	Hossz	Ciklusszám
BPL rr	10	2	2 + 2

BRK – Kényszerített megszakítás

megszakítás (MONITOR)			...B.1..
Utasítás	Kód	Hossz	Ciklusszám
BRK	00	1	7

BVC – Ugrás, ha nincs belső túlcsordulás

ugrás, ha V = 0			....
Utasítás	Kód	Hossz	Ciklusszám
BVC rr	50	2	2 + 2

BVS – Ugrás, ha van belső túlcsordulás

ugrás, ha V = 1			....
Utasítás	Kód	Hossz	Ciklusszám
BVS rr	70	2	2 + 2

CLC – CARRY törlése

C = 0			C
Utasítás	Kód	Hossz	Ciklusszám
CLC	18	1	2

CLD – Decimális mód törlése

D = 0			....D...
Utasítás	Kód	Hossz	Ciklusszám
CLD	D8	1	2

CLI – Megszakítás engedélyezése

I = 0			.....I..
Utasítás	Kód	Hossz	Ciklusszám
CLI	58	1	2

CLV – Túlcsordulást jelző bit törlése

V = 0			.V......
Utasítás	Kód	Hossz	Ciklusszám
CLV	B8	1	2

CMP – Összehasonlítás az akkumulátorral

SR A - M szerint			N.....ZC
Utasítás	Kód	Hossz	Ciklusszám
CMP #ee	C9	2	2
CMP cc	C5	2	3
CMP cc,X	D5	2	4
CMP cccc	CD	3	4
CMP cccc,X	DD	3	4 + 1
CMP cccc,Y	D9	3	4 + 1
CMP (cc,X)	C1	2	6
CMP (cc),Y	D1	2	5 + 1

CPX – Összehasonlítás az X-regiszterrel

SR X - M szerint			N.....ZC
Utasítás	Kód	Hossz	Ciklusszám
CPX #ee	E0	2	2
CPX cc	E4	2	3
CPX cccc	EC	3	4

CPY – Összehasonlítás az Y-regiszterrel

SR Y - M szerint			N.....ZC
Utasítás	Kód	Hossz	Ciklusszám
CPY #ee	C0	2	2
CPY cc	C4	2	3
CPY cccc	CC	3	4

DEC – Tártartalom csökkentése 1-gyel

M = M - 1			N.....Z.
Utasítás	Kód	Hossz	Ciklusszám
DEC cc	C6	2	5
DEC cc,X	D6	2	6
DEC cccc	CE	3	6
DEC cccc,X	DE	3	7

DEX – X-regiszter csökkentése 1-gyel

X = X - 1			N.....Z.
Utasítás	Kód	Hossz	Ciklusszám
DEX	CA	1	2

DEY – Y-regiszter csökkentése 1-gyel

Y = Y - 1			N.....Z.
Utasítás	Kód	Hossz	Ciklusszám
DEY	88	1	2

EOR – Logikai KIZÁRÓ-VAGY az akkumulátor és a tár között

A = A EOR M			N.....Z.
Utasítás	Kód	Hossz	Ciklusszám
EOR #ee	49	2	2
EOR cc	45	2	3
EOR cc,X	55	2	4
EOR cccc	4D	3	4
EOR cccc,X	5D	3	4 + 1
EOR cccc,Y	59	3	4 + 1
EOR (cc,X)	41	2	6
EOR (cc),Y	51	2	5 + 1

INC – Tártartalom növelése 1-gyel

M = M + 1 N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
INC cc	E6	2	5
INC cc,X	F6	2	6
INC cccc	EE	3	6
INC cccc,X	FE	3	7

INX – X-regiszter növelése 1-gyel

X = X + 1 N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
INX	E8	1	2

INY – Y-regiszter növelése 1-gyel

Y = Y + 1 N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
INY	C8	1	2

JMP – Feltétel nélküli ugrás

PC = cím ....

Utasítás	Kód	Hossz	Ciklusszám
JMP cccc	4C	3	3
JMP (cccc)	6C	3	5

JSR – Feltétel nélküli ugrás szubrutinba

verem = PC + 2; PC = cím ....

utasítás	Kód	Hossz	Ciklusszám
JSR cccc	20	3	6

LDA – Beolvasás az akkumulátorba

A = M N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
LDA #ee	A9	2	2
LDA cc	A5	2	3
LDA cc,X	B5	2	4
LDA cccc	AD	3	4
LDA cccc,X	BD	3	4 + 1
LDA cccc,Y	B9	3	4 + 1
LDA (cc,X)	A1	2	6
LDA (cc),Y	B1	2	5 + 1

LDX – Beolvasás az X-regiszterbe

X = M N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
LDX #ee	A2	2	2
LDX cc	A6	2	3
LDX cc,Y	B6	2	4
LDX cccc	AE	3	4
LDX cccc,Y	BE	3	4 + 1

LDY – Beolvasás az Y-regiszterbe

Y = M N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
LDY #ee	A0	2	2
LDY cc	A4	2	3
LDY cc,X	B4	2	4
LDY cccc	AC	3	4
LDY cccc,X	BC	3	4 + 1

LSR – Bitenkénti léptetés jobbra

0 → 76543210 → C   N.....ZC

Utasítás	Kód	Hossz	Ciklusszám
LSR	4A	1	2
LSR cc	46	2	5
LSR cc,X	56	2	6
LSR cccc	4E	3	6
LSR cccc,X	5E	3	7

NOP – üres utasítás

nincs   ....

Utasítás	Kód	Hossz	Ciklusszám
NOP	EA	1	2

ORA – Logikai VAGY a tár és az akkumulátor között

A = A ORA M   N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
ORA #ee	09	2	2
ORA cc	05	2	3
ORA cc,X	15	2	4
ORA cccc	0D	3	4
ORA cccc,X	1D	3	4 + 1
ORA cccc,Y	19	3	4 + 1
ORA (cc,X)	01	2	6
ORA (cc),Y	11	2	5

PHA – Akkumulátor írása a verembe

verem = A; SP = SP - 1   ....

Utasítás	Kód	Hossz	Ciklusszám
PHA	48	1	3

PHP – Feltételregiszter írása a verembe

verem = SR; SP = SP - 1   ....

Utasítás	Kód	Hossz	Ciklusszám
PHP	08	1	3

PLA – Akkumulátor olvasása a veremből

A = verem; SP = SP + 1 N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
PLA	68	1	4

PLP – Feltételregiszter olvasása a veremből

SR = verem; SP = SP + 1 NV1BDIZC

Utasítás	Kód	Hossz	Ciklusszám
PLP	28	1	4

ROL – Ciklikus léptetés balra

C ← 76543210 ← C N.....ZC

Utasítás	Kód	Hossz	Ciklusszám
ROL	2A	1	2
ROL cc	26	2	5
ROL cc,X	36	2	6
ROL cccc	2E	3	6
ROL cccc,X	3E	3	7

ROR – Ciklikus léptetés jobbra

C → 76543210 → C N.....ZC

Utasítás	Kód	Hossz	Ciklusszám
ROR	6A	1	2
ROR cc	66	2	5
ROR cc,X	76	2	6
ROR cccc	6E	3	6
ROR cccc,X	7E	3	7

RTI – Visszatérés megszakításból

PC = verem; SR = verem NV1BDIZC

Utasítás	Kód	Hossz	Ciklusszám
RTI	40	1	6

RTS – Visszatérés szubrutinból

PC = verem; PC = PC + 1 ....

Utasítás	Kód	Hossz	Ciklusszám
RTS	60	1	6

SBC – Kivonás CARRY-vel

A = A - M - (1 - C) NV....ZC

Utasítás	Kód	Hossz	Ciklusszám
SBC #ee	E9	2	2
SBC cc	E5	2	3
SBC cc,X	F5	2	4
SBC cccc	ED	3	4
SBC cccc,X	FD	3	4 + 1
SBC cccc,Y	F9	3	4 + 1
SBC (cc,X)	E1	2	6
SBC (cc),Y	F1	2	5 + 1

SEC –C ARRY 1-re állítása

C = 1 C

Utasítás	Kód	Hossz	Ciklusszám
SEC	38	1	2

SED – Decimális mód bekapcsolása.

D = 1 ....D...

Utasítás	Kód	Hossz	Ciklusszám
SED	F8	1	2

SEI – Megszakítás letiltása

I = 1 .....I..

Utasítás	Kód	Hossz	Ciklusszám
SEI	78	1	2

STA – Akkumulátor kiírása a tárba

M = A ....

Utasítás	Kód	Hossz	Ciklusszám
STA cc	85	2	3
STA cc,X	95	2	4
STA cccc	8D	3	4
STA cccc,X	9D	3	5
STA cccc,Y	99	3	5
STA (cc,X)	81	2	6
STA (cc),Y	91	2	6

STX – X-regiszter kiírása a tárba

M = X ....

Utasítás	Kód	Hossz	Ciklusszám
STX cc	86	2	3
STX cc,Y	96	2	4
STX cccc	8E	3	4

STY – Y-regiszter kiírása a tárba

M = Y ....

Utasítás	Kód	Hossz	Ciklusszám
STY cc	84	2	3
STY cc,X	94	2	4
STY cccc	8C	3	4

TAX – Akkumulátor áttöltése az X-regiszterbe

X = A N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
TAX	AA	1	2

TAY – Akkumulátor áttöltése az Y-regiszterbe

Y = A N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
TAY	A8	1	2

TSX – Veremmutató áttöltése az X-regiszterbe

X = SP   N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
TSX	BA	1	2

TXA – X-regiszter áttöltése az akkumulátorba

A = X   N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
TXA	8A	1	2

TXS – X-regiszter áttöltése a veremmutatóba

SP = X   ....

Utasítás	Kód	Hossz	Ciklusszám
TXS	9A	1	2

TYA – Y-regiszter áttöltése az akkumulátorba

A = Y   N.....Z.

Utasítás	Kód	Hossz	Ciklusszám
TYA	98	1	2

Ciklusszámnál: +1 laphatár átlépésekor a ciklusszám 1-gyel nő
+2 feltétel teljesülésekor a ciklusszám 2-vel nő

129,- Ft